UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Ingeniería del Software e Inteligencia Artificial

TESIS DOCTORAL

Desarrollo de aplicaciones XML mediante herramientas de construcción de procesadores de lenguaje

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Antonio Sarasa Cabezuelo

Director

José Luis Sierra Rodríguez

Madrid, 2013

UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA

Departamento de Ingeniería del Software e Inteligencia Artificial

DESARROLLO DE APLICACIONES XML MEDIANTE HERRAMIENTAS DE CONSTRUCCIÓN DE PROCESADORES DE LENGUAJE

TESIS DOCTORAL

Presentada por:
Antonio Sarasa Cabezuelo

Bajo la dirección del Doctor:
José Luis Sierra Rodríguez

Madrid, 2012

DESARROLLO DE APLICACIONES XML MEDIANTE HERRAMIENTAS DE CONSTRUCCIÓN DE PROCESADORES DE LENGUAJE

Memoria que presenta para optar al título de Doctor en Informática:
Antonio Sarasa Cabezuelo

Bajo la dirección del Doctor:
José Luis Sierra Rodríguez

Universidad Complutense de Madrid
Facultad de Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Madrid, 2012

A mis Hijos, a mis Padres y a mis Hermanas.
A Mamen.

Agradecimientos

En primer lugar quiero agradecer muy especialmente a mi mentor, director de tesis y amigo José Luis Sierra, pues sin él no existiría esta tesis doctoral. Son innumerables todas las cosas que he aprendido de él, y por las que le tengo que estar agradecido. Sin embargo, me gustaría destacar cuatro. En primer lugar quiero agradecerle que confiara en mí y me propusiera realizar este trabajo de investigación. En segundo lugar quiero agradecerle todo el tiempo y esfuerzo que ha dedicado a la consecución exitosa de este trabajo. En tercer lugar quiero agradecerle el apoyo que he recibido siempre tanto en el ámbito académico como personal. Y en cuarto lugar quiero agradecerle todo lo que he aprendido de él sobre lo que es investigar y trabajar en la Universidad. Para mi ha sido y es un punto de referencia de lo que debe ser un investigador, un científico y un profesor de universidad. Me siento muy orgulloso y afortunado de haber sido su doctorando y de poder trabajar con él. Muchas Gracias José Luis.

En segundo lugar quiero agradecer a todas las personas que han compartido su vida conmigo durante estos años de trabajo de tesis y que han soportado mis incertidumbres, mis dudas, mis horas de trabajo, mis alegrías, mis penas,…Todos ellos han contribuido de alguna manera a la realización de este trabajo. Aunque no puedo nombrar a todos de forma explícita, no quiero dejar de mencionar a algunos de ellos.

A mis compañeros del Grupo de Investigación ILSA. En particular quiero hacer una mención explícita a Bryan Temprado y a Daniel Rodríguez, por su apoyo y participación en la obtención de algunos de los resultados de esta tesis, y a Alfredo Fernández-Valmayor por su participación inicial en el trabajo de investigación de esta tesis. También quiero agradecer su apoyo a José María Ruiz y a Ángel Luis Encinas.

A todos mis compañeros de la Universidad Complutense de Madrid con los que he compartido estos años de trabajo. En particular no puedo dejar de mencionar a Mercedes Gómez Albarrán, Rubén Fuentes, Antonio Navarro, Ana Fernández-Pampillón y Sonia Estévez.

A mi compañera y amiga de la UNED, Covadonga Rodrigo San Juan. Muchas Gracias por tu apoyo y ánimo constantes a lo largo de estos años.

A mis compañeros de Red.es. En particular quiero agradecer su apoyo incansable y constante a Manuel Canabal, a Juan Carlos Sacristán y a Juan Ramón González-Puyol. Muchas Gracias por vuestra paciencia y por todo lo que he aprendido de vosotros.

A mis amigos David, Alberto, Virginia, María Jesús, Arturo y Javi. Muchas Gracias por vuestro apoyo incondicional en todos los momentos de mi vida.

A mis amigos de Motril Carlos y María. Gracias por vuestro apoyo y por tratarme como a uno más de vuestra familia.

A Javier Alvarado y a todos los demás por haberme aceptado entre vosotros.

Así mismo quiero agradecer a toda mi familia, en especial a mis padres Jesús y Loli, a mis hermanas Estibaliz, Begoña y Bárbara, a mis hijos Lucía y Antonio, y a mi

prima Mercedes por el apoyo, cariño y confianza que han demostrado durante todos estos años. También deseo agradecer a Mamen su comprensión, su cariño, y su apoyo diario sin el que sin duda no habría finalizado esta tesis. Muchas Gracias. Os quiero a todos.

Por último quiero hacer mención a dos personas con las que me habría gustado compartir este momento: mi abuelo Jesús y mi abuela Joaquina. Por vosotros.

Antonio Sarasa Cabezuelo

Madrid, 11 de Agosto de 2012

Acerca de este documento

Este trabajo es presentado como una recopilación de publicaciones, de acuerdo a la sección 4.4 de la Normativa de desarrollo del Real Decreto 1393/2007, de 29 de Octubre, por el que se establece la ordenación de las enseñanzas universitarias oficiales de la Universidad Complutense de Madrid (Aprobada por el Consejo de Gobierno a 14 de Octubre de 2008, modificado por la Comisión Permanente del Consejo de Gobierno con fecha de 29 de Octubre de 2010, publicado en el BOUC el 20 de Noviembre de 2008).

Los artículos presentados son los siguientes:

- Sarasa-Cabezuelo, A., Navarro-Iborra, A., Sierra-Rodríguez, J.L, Fernández-Valmayor, A. Building a Syntax Directed Processing Environment for XML Documents by Combining SAX and JavaCC. 3rd *International Workshop on XML Data Management Tools & Techniques* (XANTEC '08) - *19th International Conference on Database and Expert Systems Application* (DEXA'08), 1-5 de Septiembre de 2008, Turin, Italia. DEXA'08 Workshops. IEEE Computer Society, pp.256-260
- Sarasa-Cabezuelo, A., Temprado-Battad, B., Sierra-Rodríguez, J.L, Fernández-Valmayor, A. XML Language-Oriented Processing with XLOP. *5th International Symposium on Web and Mobile Information Services* (WAMIS'09) - 23rd *International Conference on Advanced Information Networking and Applications* (AINA'09), 26-29 de Mayo de 2009, Bradford, Inglaterra. AINA'09 Workshops. IEEE Computer Society, pp. 322-327
- Sarasa-Cabezuelo, A., Sierra-Rodriguez, J.L, Fernández-Valmayor, A. Processing Learning Objects with Attribute Grammars. 9th *IEEE International Conference on Advanced Learning Technologies* (ICALT 2009). 15-17 de Julio de 2009, Riga, Letonia. IEEE Computer Society, pp. 527-531.
- Sarasa-Cabezuelo, A, Sierra-Rodríguez, J.L, Fernández-Valmayor, A. Procesamiento de Documentos XML Dirigido por Lenguajes en Entornos de E-Learning. IEEE RITA, Agosto de 2009. Volumen 4. Numero 3. pp: 175-183. ISSN 1932-8540[1].
- Sarasa-Cabezuelo, A., Martínez-Avilés, A., Sierra-Rodríguez, J.L., Fernández-Valmayor, A. A Generative Approach to the Construction of Application-Specific XML Processing Components. *35th Euromicro Software Engineering and Advanced Applications Conference* (SEAA'09). 27-29 de Agosto de 2009, Patras, Grecia. IEEE Computer Society, pp. 345-352
- Sarasa-Cabezuelo, A., Temprado-Battad, B., Martínez-Avilés, A., Sierra-Rodriguez, J.L., Fernández-Valmayor, A. Building an Enhanced Syntax-Directed Processing Environment for XML Documents by Combining StAX and CUP. 4th *International Workshop on Flexible Database and Information Systems Technology* (FlexDBIST-09) - *19th International Conference on Database and Expert Systems Application* (DEXA'09). 31 de Agosto a 4 de Septiembre de 2009, Linz, Austria. DEXA'09 Workshops. IEEE Computer Society, pp.427-431

[1] Versión extendida y revisada del trabajo (Sarasa et al., 2008b), presentado en SIIE'08 y seleccionado para formar parte de un número especial sobre desarrollo de sistemas educativos en IEEE RITA.

- Sarasa-Cabezuelo, A., Temprado-Battad, B., Sierra-Rodríguez, J.L. Engineering web services with attribute grammars: a case study. *ACM SIGSOFT Software Engineering Notes*, 24 de Enero de 2011. Volumen 36. Número 1. pp: 1-8. DOI: 10.1145/1921532.1921545.
- Sarasa-Cabezuelo, A., Temprado-Battad, B., Rodriguez-Cerezo, D., Sierra-Rodríguez, J.L. Building XML-Driven Application Generators with Compiler Construction Tools. *Computer Science and Information Systems.* Junio de 2012. Volumen 9, Número 2. pp: 485-504. ISSN: 1820-0214 (Índice de impacto JCR en 2011: 0.625)
- Sarasa-Cabezuelo, A., Sierra-Rodríguez, J.L. The Grammatical Approach: A Syntax-Directed Declarative Specification Method for XML Processing Tasks. *Computer Standards & Interfaces. En prensa* (versión on-line 9 de Julio 2012). DOI: 10.1016/j.csi.2012.06.006. ISSN: 0920-5489. (Índice de impacto JCR en 2011: 1.255)

De acuerdo a la normativa, este documento también incluye una introducción y un estudio del estado del arte del dominio. También hay una descripción de los objetivos propuestos para este trabajo y una discusión integrando los contenidos de los 9 artículos y relacionándolos con los objetivos mencionados. Adicionalmente, se presenta una sección con el objetivo de analizar los resultados, así como de resumir unas conclusiones y trabajo futuro. Finalmente, se incluye una bibliografía que integra y complementa todas las referencias de los artículos incluidos.

Resumen

XML (eXtensible Markup Language) es una especificación propuesta por el Word Wide Web Consortium (W3C) para la definición de lenguajes de marcado. Este tipo de lenguajes permite definir documentos electrónicos que se caracterizan porque la estructura de la información contenida en el documento se hace explícita mediante el uso de marcas o etiquetas debidamente anidadas. Esta tecnología resulta especialmente interesante, dado que actualmente muchos de los desarrollos de software que requieren gestionar información, utilizan XML para definir documentos que almacenan la información que se intercambian los componentes de los sistemas informáticos. Un rasgo característico del marcado XML lo constituye la separación explicita entre estructura y procesamiento. En este sentido, el marcado plasma la estructura lógica de la información pero no permite especificar las posibles formas de procesarla. Así, para abordar el problema del procesamiento de documentos XML se han propuesto múltiples tecnologías, que pueden clasificarse en tecnologías específicas, centradas en implementar tareas de procesamiento muy específicas, y tecnologías de propósito general, aplicables a cualquier tarea de procesamiento. En ambos casos, la implementación del procesamiento requiere la programación explicita del mismo en un lenguaje de procesamiento especifico en el primer caso, o en un lenguaje de programación de propósito general en el segundo caso.

Esta Tesis ahonda en un enfoque diferente a los especificados anteriormente para el desarrollo de aplicaciones que procesan documentos XML. Este enfoque se basa en la naturaleza lingüística de los lenguajes de marcado, y en el uso de técnicas tradicionales de procesamiento de lenguajes de programación. La especificación sintáctica de un lenguaje de marcado (es decir, cómo se construye el lenguaje y cómo se usan las marcas que permiten crear un documento XML) puede resolverse desde una perspectiva lingüística mediante la definición de una gramática formal que establece la sintaxis del lenguaje. La idea esencial propuesta en esta Tesis consiste en abordar el problema del procesamiento de un lenguaje de marcado siguiendo también un enfoque lingüístico, de una manera similar al problema sintáctico. Dicho enfoque propugna el diseño de gramáticas formales apropiadas para los lenguajes de marcado, equivalentes a las utilizadas en su definición, pero específicamente orientadas a cada tarea de procesamiento. Una vez que se dispone de dichas gramáticas, el enfoque promueve plantear el desarrollo de las aplicaciones de procesamiento como el desarrollo de procesadores para los lenguajes descritos por las gramáticas, pudiendo emplearse, para ello, las mismas herramientas que se utilizan para desarrollar procesadores de lenguajes de programación y de otros lenguajes informáticos.

En el trabajo de Tesis se propone el uso de técnicas clásicas para el desarrollo de procesadores de lenguaje informáticos en la realización práctica del enfoque lingüístico: (i) herramientas de generación de traductores convencionales basados en esquemas de traducción, y (ii) gramáticas de atributos. Como resultado se obtienen métodos sistemáticos de desarrollo de aplicaciones de procesamiento de documentos XML, que conciben el proceso de desarrollo como el de un procesador para un lenguaje informático específico. Este método de desarrollo tiene dos ventajas clave para el desarrollador: por una parte (i) el alto grado de madurez

alcanzado por las técnica de desarrollo de traductores para lenguajes informáticos y las numerosas herramientas software existentes que soportan el desarrollo de dichos componentes, y por otra parte (ii) la facilidad de mantenimiento de la aplicación de procesamiento desarrollada, ya que dicho procesamiento se especifica como extensiones apropiadas de las gramáticas para los lenguajes de marcado utilizados.

La Tesis muestra la factibilidad del enfoque lingüístico mostrando, primeramente, cómo utilizar generadores de traductores convencionales (e.g., YACC, JavaCC, ANTLR,…) en la construcción de aplicaciones de procesamiento de documentos XML. Así mismo, la Tesis muestra cómo facilitar las especificaciones dirigidas por sintaxis de dichas aplicaciones utilizando gramáticas de atributos, y cómo soportar automáticamente la traducción de tales especificaciones a implementaciones eficientes. Para ello, como parte del trabajo de la Tesis se ha implementado un entorno denominado XLOP (XML *Language Oriented Processing*), que tomando como entrada la especificación de una aplicación de procesamiento de documentos XML en forma de gramática de atributos y un conjunto de componentes software, automatiza el desarrollo de la citada aplicación.

Por último la memoria de la Tesis describe la aplicabilidad de los resultados de la misma en diferentes escenarios de uso en los dominios de las aplicaciones intensivas en contenidos y del aprendizaje electrónico (e-Learning), facilitando, de esta forma, casos de estudio para aquellos desarrolladores que opten por implementar este tipo de aplicaciones de procesamiento usando la propuesta lingüística descrita en esta Tesis.

Estructura del Trabajo

El núcleo de este trabajo es una recopilación de publicaciones editadas, además de un bloque previo que integra y comenta las contribuciones de cada uno de los artículos.

Por tanto, la estructura de este trabajo es como sigue:

–Capítulo 1. Introducción.

–Capítulo 2. Estado del dominio.

–Capítulo 3. Antecedentes, Objetivos y Planteamiento del trabajo.

–Capítulo 4. Discusión de las Contribuciones de los Artículos.

–Capítulo 5. Conclusiones y Trabajo futuro.

–Capítulo 6. Artículos Presentados.

Índice general

Capítulo 1: Introducción

1.1 Motivación de la Investigación

XML (eXtensible Markup Language) (Bray et al., 2008) es un metalenguaje propuesto por el Word Wide Web Consortium (W3C) para la definición de lenguajes de marcado, que tiene su origen en el estándar SGML (Standard Generalized Markup Language) (Goldfarb, 1991), propuesto por la International Organization for Standarization (ISO). Los lenguajes creados con XML se caracterizan por introducir un conjunto de etiquetas o marcas que permiten describir la estructura lógica de la información contenida en documentos electrónicos. En la última década, los lenguajes de marcado han alcanzado una amplia difusión e importancia debido al uso extendido que se ha realizado de los documentos XML como mecanismo de intercambio de información entre los componentes de los sistemas informáticos. En este escenario, los documentos XML encapsulan la información que reciben o que generan los componentes. La ventaja que proporciona el lenguaje de marcado para estos fines es que permite fijar cómo va a ser estructurada la información en los documentos XML, de forma que dicha información sea legible por humanos y procesable por programas.

Un rasgo característico de los lenguajes de marcado lo constituye el hecho de que las marcas que definen estos lenguajes para estructurar la información no permiten especificar cómo se debe procesar dicha información. Debido a ello, el marcado produce una separación explicita de la especificación de la estructura de la información con respecto a la especificación de cómo se debe procesar la información (Coombs et al., 1987). XML no define mecanismos que normen cómo llevar a cabo dichos aspectos de procesamiento.

La carencia en XML para especificar el procesamiento de la información ha llevado a la definición de mecanismos externos a XML que permiten definir cómo se debe procesar la información contenida en un documento XML. Entre las soluciones más ampliamente adoptadas existen dos enfoques diferentes:

- El primer enfoque se caracteriza por la definición de tecnologías que permiten especificar tipos de procesamiento muy específicos, tales como llevar a cabo consultas sobre la información contenida en el documento o la transformación de documentos XML de un tipo a otro. Este enfoque resulta especialmente ventajoso cuando se puede encontrar una tecnología específica que se ajusta al procesamiento que se quiere llevar a cabo. Sin embargo cuando la tarea a

realizar es sumamente compleja o específica, probablemente no exista una tecnología específica para cubrir este procesamiento.

- El segundo enfoque trata de resolver el problema de limitación expresiva propio de las tecnologías específicas, definiendo, para ello, tecnologías de propósito general, capaces de abordar cualquier tarea de procesamiento. Las tecnologías representativas de este enfoque se caracterizan porque se presentan como un conjunto de herramientas embebidas dentro de un lenguaje de programación de propósito general, tal como Java o C# (entre ellas, *parsers* XML que permiten leer documentos XML y comprobar el correcto uso del marcado, y APIs que facilitan un conjunto de operaciones para acceder y manipular la información contenida en dichos documentos). El procesamiento se programa utilizando el lenguaje de programación genérico en el que se embebe esta tecnología, desde el que se hace uso de estas herramientas para acceder a la información y así poder procesarla. La principal ventaja de este enfoque es que puede aplicarse a cualquier tarea de procesamiento de documentos XML, aunque su principal desventaja se debe a la dificultad de uso propia del manejo de marcos de aplicación sofisticados a través de lenguajes de propósito general.

Aparte de las diferencias obvias indicadas, ambos enfoques comparten el hecho de que los documentos XML son tratados como *estructuras de datos* (en unos casos como árboles y en otros como secuencias de eventos o bien como ristras de elementos de información). Sin embargo *en ningún caso se explota el sustrato lingüístico presente en los lenguajes de marcado*. Efectivamente, un lenguaje de marcado es esencialmente un tipo de lenguaje formal para el que se puede definir una gramática documental que lo describe. Así, implementar una aplicación que realiza un procesamiento sobre un tipo de documento XML puede interpretarse como la implementación de una aplicación que tiene por objeto el procesamiento de un lenguaje: el lenguaje de marcado. Por tanto realmente la aplicación que se implementa es un procesador del lenguaje de marcado que define el tipo de documento XML. Esta visión lingüística de la aplicación de procesamiento como un procesador de un lenguaje constituye la hipótesis básica de esta Tesis doctoral.

Esta Tesis ha sido desarrollada en el seno del grupo de investigación ILSA (Ingeniería de Lenguajes Software y Aplicaciones)[1], grupo oficialmente reconocido por la UCM número 962022. A continuación se exponen los objetivos y planteamiento de la línea de investigación de la misma.

1.2 Objetivos y Planteamiento de la Línea de Investigación

Tal como se ha propuesto anteriormente, el desarrollo de una aplicación que implementa un procesamiento sobre documentos XML puede plantearse en términos del desarrollo de un procesador del lenguaje de marcado XML que define el tipo de documento XML a procesar. Este enfoque lingüístico presenta como ventajas principales:

[1] http://ilsa.fdi.ucm.es

- La posibilidad de aprovechar el conocimiento, métodos, técnicas y herramientas disponibles en un dominio tan maduro y estable como es el de la construcción de procesadores de lenguajes.
- Plantear un método sistemático de construcción de aplicaciones de procesamiento de documentos XML que aproveche dicha experiencia acumulada. Esto será especialmente relevante a la hora de abordar lenguajes de marcado complejos y/o dotados de estructuras sofisticadas.

De esta forma, la investigación planteada en este trabajo de Tesis consiste en demostrar la viabilidad del desarrollo de aplicaciones de procesamiento de documentos XML como procesadores de lenguajes, su descripción a partir de *gramáticas de atributos* (Knuth, 1968), y su generación de manera (semi)automática a partir de éstas últimas. Para ello se plantean los siguientes objetivos:

- El primer objetivo se centra en demostrar que es posible construir aplicaciones de procesamiento de documentos XML combinando herramientas de generación de analizadores sintácticos convencionales con marcos de trabajo convencionales para el procesamiento XML.
- El segundo objetivo se centra en demostrar que es posible especificar declarativamente tareas de procesamiento de documentos XML utilizando gramáticas de atributos, y que es posible generar las aplicaciones que implementan el procesamiento descrito partiendo de estas especificaciones de alto nivel.

El primer objetivo se ha abordado mediante la definición de un método general para el desarrollo orientado a lenguajes de componentes de procesamiento XML dirigidos por la sintaxis (Sarasa et al., 2012a). Utilizando este método es posible construir entornos sofisticados de procesamiento XML que combinan herramientas de generación de analizadores convencionales con marcos de trabajo de procesamiento XML de propósito general orientados a flujos. Este método ha sido puesto en práctica desarrollando dos entornos, uno basado en SAX y JavaCC (Sarasa et al., 2008a), y otro basado en StAX y CUP (Sarasa et al., 2009e). Este método se caracteriza porque aprovecha, de las herramientas de generación de traductores, la posibilidad de organizar el procesamiento específico de la aplicación en términos lingüísticos, y aprovecha, de los marcos de trabajo de procesamiento XML, las características de uso de propósito general que aparecen en cualquier aplicación de procesamiento XML. Aunque existen propuestas similares inspiradas en la misma idea de utilizar generadores de traductores para construir aplicaciones de procesamiento XML (se analizan en el capítulo 2), tales propuestas se diferencian con respecto al método propuesto en esta Tesis en que se apoyan en el uso de lenguajes de especificación dedicados al dominio del procesamiento de documentos XML, en lugar de en herramientas de generación de traductores convencionales, como ocurre con nuestra propuesta.

El segundo objetivo ha supuesto, primeramente, la definición de un método general para la especificación de tareas de procesamiento de documentos XML (Sarasa et al., 2012b) basado en el formalismo de las gramáticas de atributos. El método describe cómo se pueden obtener gramáticas incontextuales para los lenguajes de marcado que definen los documentos XML y cómo se pueden modelar las tareas de procesamiento sobre los documentos XML mediante

gramáticas de atributos. Existen propuestas similares que usan las gramáticas de atributos para caracterizar ciertas tareas de procesamiento de documentos XML (se analizan también en el capítulo 2), pero se diferencian de la propuesta en esta Tesis en el fuerte acoplamiento que presentan con las gramáticas documentales; por el contrario, nuestra propuesta entiende la formulación de gramáticas específicas de cada tarea de procesamiento particular como un elemento central y característico del proceso de desarrollo. El método se ha concretado en la implementación de un entorno de desarrollo de aplicaciones de procesamiento de documentos XML denominado *XLOP* (XML *Language-Oriented Processing*) (Sarasa et al., 2009a) basado en el método gramatical de especificación y en el método lingüístico propuesto. El entorno toma como entrada especificaciones de procesamiento de documentos XML basadas en gramáticas de atributos, las transforma en esquemas de traducción para la herramienta de generación de analizadores CUP, y genera semiautomáticamente a partir de estos últimos las aplicaciones que implementan los procesamientos descritos en las especificaciones de la entrada.

Para validar los resultados obtenidos en la consecución de cada objetivo se han desarrollado varios casos de estudio:

- Generación de aplicaciones de búsqueda de caminos mínimos en redes de metro (Sarasa et al., 2012a). Este caso de estudio, ya utilizado en (Sierra et al., 2004) (Sierra et al., 2006b) para validar propuestas previas de desarrollo de aplicaciones basado en tecnologías XML, permite generar aplicaciones que informan sobre rutas de mínimo coste en redes de metro a partir de la descripción de sus principales aspectos (estructura y dinámica de la red, así como otros aspectos relativos a la interfaz de usuario) en forma de documentos XML.
- Sistema para la producción de tutores socráticos (Sarasa et al., 2009c) (Sarasa et al., 2009d). Este caso de estudio, también utilizado en (Sierra et al., 2008a) como mecanismo de validación de propuestas de desarrollo de aplicaciones basado en XML, así como en (Sierra et al., 2007b) (Sierra et al., 2007c) (Sierra et al., 2008d) como caso de estudio en el desarrollo de aplicaciones dirigido por lenguajes, permite generar tutores socráticos clásicos, del estilo de los descritos en (Bork, 1985), a partir de su descripción como documentos XML.
- Comprobación de restricciones sobre metadatos de objetos de aprendizaje (Sarasa et al., 2009b) (Sarasa et al., 2009c) (Sarasa et al., 2011) (Sarasa et al., 2012b). Este caso de estudio se centra en el modelo de objetos de aprendizaje manejados por repositorios tipo *Chasqui*, un sistema para la construcción de colecciones de objetos de aprendizaje en dominios especializados (Sierra et al., 2006a), y aborda el desarrollo de un servicio para la especificación y comprobación de restricciones sobre los contenidos de los metadatos de dichos objetos.

Capítulo 2: Estado del Dominio

Como se ha indicado en el capítulo anterior, esta Tesis aborda la problemática relativa a la implementación de aplicaciones que procesan documentos XML. Para ello se propone abordar su desarrollo desde una perspectiva lingüística y enfocarlo como la construcción de un procesador de lenguaje en el que las tareas de procesamiento se encuentran descritas en especificaciones de alto nivel. De esta forma la investigación realizada se apoya por una parte en las herramientas de construcción de procesadores de lenguaje, y por otra parte en las técnicas de especificación de dichos procesadores (más concretamente, las basadas en *gramáticas de atributos*). Así mismo, en esta investigación se han tenido en cuenta los trabajos existentes que de forma similar implementan el procesamiento de documentos XML mediante soluciones dirigidas por lenguajes (es decir, entendiendo el desarrollo de aplicaciones de procesamiento de documentos XML como el diseño e implementación de procesadores para los correspondientes lenguajes de marcado). En este capítulo se analizan todos estos aspectos, a fin de contextualizar adecuadamente el trabajo de Tesis presentado.

La estructura del capítulo es como sigue: el capítulo comienza revisando los conceptos básicos acerca de los documentos XML y los lenguajes de marcado, así cómo las tecnologías que se han desarrollado para realizar su procesamiento. Seguidamente se analizan las tecnologías de construcción de procesadores de lenguaje. A continuación se analizan las técnicas de especificación semántica basadas en gramáticas de atributos. Por último se realiza un estudio de los principales enfoques dirigidos por lenguajes al procesamiento de documentos XML. Para finalizar se presentan las conclusiones más relevantes que resultan de este estudio.

2.1 Representación de Documentos Electrónicos con XML

En este apartado se presentan los conceptos básicos relativos al uso de XML para la representación de documentos electrónicos. Para ello se introducen las generalidades del lenguaje, las extensiones orientadas a su modularización, y el concepto de *gramática documental*, que permite definir declarativamente vocabularios de marcado para aplicaciones específicas del lenguaje.

2.1.1 Marcado Básico de Documentos con XML

XML (*eXtensible Markup Language*) (Bray et al., 2008) es un lenguaje de marcado generalizado propuesto por el World Wide Web Consortium (W3C), basado en el estándar SGML (*Standard Generalized Markup Language*) (Goldfarb, 1991), que permite definir y utilizar *lenguajes de marcas* para poder representar documentos electrónicos.

Los lenguajes de marcas norman cómo utilizar *marcas* para representar documentos electrónicos. Una *marca* es cualquier elemento de un documento electrónico que proporciona la metainformación necesaria para describir la estructura o el procesamiento de los contenidos del documento. En XML, las marcas más habituales son pares de *etiquetas* encerradas entre corchetes angulares de la forma <*etiqueta*> y </*etiqueta*> que delimitan el comienzo y final del contenido del documento al cual se refieren. Dichas marcas son *descriptivas*, ya que permiten explicitar la estructura del documento, en lugar de su procesamiento. En este sentido la representación de un documento mediante XML facilita la *autodocumentación* mediante un formato flexible, ya que habitualmente será posible entender la representación del documento leyendo las etiquetas.

```
<Curso>
   <asignatura>
           <nombre> Estructuras de datos </nombre>
           <area> Software </area>
           <cuatrimestre> Primero </cuatrimestre>
   </asignatura>
   <asignatura>
           <nombre> Teoría de Autómatas y Lenguajes formales </nombre>
           <area> Ciencias de la computación </area>
           <cuatrimestre> Anual </cuatrimestre>
   </asignatura>
   <Profesor>
           <nombre> Peter Strauss </nombre>
           <Departamento> Ciencias de la Computación </Departamento>
           <Facultad> Informática </Facultad>
   </Profesor>
</Curso>
```

Figura 2.1. Ejemplo de documento XML.

La estructura básica de un documento XML es el *elemento*, que se define como un par de etiquetas de inicio y finalización coincidentes y el contenido que aparece entre ellas. En particular, en los documentos XML siempre existe un único elemento raíz que abarca al resto de elementos del documento. A modo de ejemplo, en la Figura 2.1 se representa un ejemplo de documento XML, en el que se puede observar que tiene como elemento raíz uno con la etiqueta `<Curso>`. Dicho ejemplo también evidencia cómo entre el contenido de un elemento pueden aparecer otros elementos, dando lugar al *anidamiento* de elementos, con la única restricción de que todos los elementos se aniden adecuadamente[1]. En el ejemplo anterior se puede observar que los elementos `<nombre>`, `<area>`, y `<cuatrimestre>` se encuentran anidados dentro del elemento `<asignatura>`.

Con el fin de matizar la descripción de la estructura de los documentos, en los elementos XML pueden aparecer *atributos*, que son cadenas de la forma *nombre*=*valor* que se localizan en el interior de la etiqueta de inicio de un elemento

[1] El contenido que aparece entre las etiquetas de inicio y finalización de un elemento se dice que se encuentra en el contexto del elemento, de manera que las etiquetas de los elementos están anidadas adecuadamente cuando toda etiqueta de inicio tiene una única etiqueta de finalización coincidente que se encuentra en el contexto del mismo elemento padre.

justo antes del cierre “>” de la misma, pudiendo solamente aparecer una única vez en una etiqueta dada. En el ejemplo de la Figura 2.2, se ha extendido el ejemplo de la Figura 2.1, y se han introducido los atributos `carácter` y `tipo` de las etiquetas `<asignatura>` y `<profesor>` respectivamente.

```
<Curso>
   <asignatura carácter="obligatoria">
           <nombre> Estructuras de datos </nombre>
           <area> Software </area>
           <cuatrimestre> Primero </cuatrimestre>
   </asignatura>
   <asignatura carácter="optativa">
           <nombre> Teoría de Autómatas y Lenguajes formales </nombre>
           <area> Ciencias de la computación </area>
           <cuatrimestre> Anual </cuatrimestre>
   </asignatura>
   <Profesor tipo="titular">
           <nombre> Peter Strauss </nombre>
           <Departamento> Ciencias de la Computación </Departamento>
           <Facultad> Informática </Facultad>
   </Profesor>
</Curso>
```

Figura 2.2. Ejemplo de documento XML con atributos en elementos.

2.1.2 Mecanismos de Modularización

XML no fija las etiquetas de los lenguajes de marcas, de manera que es posible que lenguajes distintos definan marcas iguales. Para permitir que las marcas representen nombres universalmente únicos se introduce un mecanismo denominado *espacio de nombres*. Dicho mecanismo no está incluido en la propia especificación XML, sino que se define en una especificación independiente (Bray et al., 2009) El mecanismo en sí concibe nombres de etiquetas formados por dos componentes diferentes:

- El *espacio de nombres* de la etiqueta, representado mediante un identificador de recurso universal (URI) (Berners-Lee et al., 2005).
- El *nombre local* de la etiqueta.

Para facilitar la aplicación del mecanismo, los espacios de nombres se asocian a abreviaturas en el elemento raíz del documento, mediante atributos de la forma `xmlns:` *Abreviatura*="*identificador de recurso universal*", especificándose los correspondientes nombres de etiquetas como *Abreviatura*: *Nombre Local*.

Un documento puede definir más de un espacio de nombres, declarado en el elemento raíz, siendo posible asociar espacios de nombres distintos a elementos diferentes. Este hecho evidencia cómo el mecanismo de los espacios de nombres facilita la combinación de distintos lenguajes de marcas para representar un mismo documento electrónico. Así mismo, de entre todos los espacios de nombres que se pueden usar en los elementos de un documento, es posible definir un espacio de nombres predeterminado[2], de manera que los elementos sin un prefijo de espacio de nombres se entienden como que pertenecen al espacio de nombres predeterminado.

La Figura 2.3 ejemplifica el mecanismo de los espacios de nombres. En dicho ejemplo se ha extendido el ejemplo de la Figura 2.2, definiéndose dos espacios de nombre: el espacio por defecto `"http://www.fdi.ucm/grado_ingenieria_software"`, y el espacio con

[2] Se declara en el elemento raíz como `xmlns="`*identificador de recurso universal*`"`.

abreviatura `doblegrado`. Además se puede observar que se han mezclado etiquetas de ambos espacios de nombres.

```
<Curso xmlns="http://www.fdi.ucm/grado_ingenieria_software"
              xmlns:doblegrado="http://www.fdi.ucm.es/doble_grado" >
   <doblegrado:asignatura carácter="obligatoria">
          <doblegrado:nombre> Estructuras de datos </doblegrado:nombre>
          <doblegrado:area> Software </doblegrado:area>
          <doblegrado:cuatrimestre> Primero </doblegrado:cuatrimestre>
   </doblegrado:asignatura>
   <asignatura carácter="optativa">
          <nombre> Teoría de Autómatas y Lenguajes formales </nombre>
          <area> Ciencias de la computación </area>
          <cuatrimestre> Anual </cuatrimestre>
   </asignatura>
   <Profesor tipo="titular">
          <nombre> Peter Strauss </nombre>
          <Departamento> Ciencias de la Computación </Departamento>
          <Facultad> Informática </Facultad>
   </Profesor>
</Curso>
```

Figura 2.3. Ejemplo de documento XML con espacios de nombres.

2.1.3 Gramáticas Documentales

Aunque la especificación XML no establece restricciones en cuanto a la utilización del marcado, sin embargo en muchos ámbitos y situaciones sí es necesario imponer ciertas restricciones. En este sentido XML permite utilizar para limitar los posibles usos del marcado un enfoque lingüístico basado en definir gramáticas formales (*gramáticas documentales*) para los lenguajes de marcas.

La propia especificación XML incluye como sub-lenguaje un lenguaje para describir un tipo de gramáticas documentales denominadas DTDs (*Document Type Definitions*). Este sub-lenguaje permite definir restricciones sobre el aspecto de los subelementos y atributos de cada elemento de un lenguaje de marcas. Las DTDs son opcionales en los documentos XML, y están formadas por una lista de reglas que describen el orden en el que los subelementos y el contenido pueden aparecer en el interior de un elemento mediante *expresiones regulares*. Dichas expresiones se denominan *modelo de contenidos* de los elementos, y pueden involucrar los siguientes operadores: | (o), + (una o más apariciones), * (cero o más apariciones), y ? (cero o una aparición). Así mismo, utilizan las siguientes palabras clave: `#PCDATA` para indicar que el elemento se trata de un dato de texto, `EMPTY` para indicar que el elemento no tiene ningún contenido, y `ANY` para indicar que no se han definido restricciones sobre los sub-elementos del elemento (cualquier elemento podría ser subelemento del elemento). En la Figura 2.4 se muestra un ejemplo de DTD para el tipo de documento ejemplificado en la Figura 2.1.

```
<!ELEMENT Curso ( (asignatura | Profesor)+)>
<!ELEMENT asignatura ( nombre, area, cuatrimestre )>
<!ELEMENT Profesor ( nombre, Departamento, Facultad )>
<!ELEMENT nombre ( #PCDATA )>
<!ELEMENT area ( #PCDATA )>
<!ELEMENT cuatrimestre ( #PCDATA )>
<!ELEMENT Departamento ( #PCDATA )>
<!ELEMENT Facultad ( #PCDATA )>
```

Figura 2.4. Ejemplo de DTD para el tipo de documento de la Figura 2.1.

En las DTDs también pueden declararse los atributos que puede tener cada elemento. Para cada atributo se especifica su nombre, su tipo, y su obligatoriedad:

– Los principales tipos contemplados por las DTDs son: `CDATA` para indicar que el atributo contiene datos de caracteres, `ID`[3] para indicar que se trata de un identificador único para el elemento, `IDREF` para indicar que se trata de una referencia a un elemento, por lo que el valor que contenga debe corresponderse con un valor que aparezca en el atributo `ID` de algún elemento en el documento, `IDREFS` para indicar que se trata de una lista de referencias separadas por espacios, `NMTOKEN` o `NMTOKENS` para indicar que el valor puede ser, respectivamente, un símbolo o secuencia de símbolos, o bien una enumeración explícita de los valores permitidos.

–Por su parte, la declaración de obligatoriedad puede ser: `#REQUIRED` para indicar que se debe especificar un valor para el atributo en cada elemento, `#IMPLIED` para indicar que no se establece ningún valor predeterminado, o bien un *valor concreto* como predeterminado, de manera que en los documentos XML para cada instancia del elemento que no tenga especificado un valor para dicho atributo, se tome como valor el declarado como predeterminado en la DTD.

En la Figura 2.5, se muestra un ejemplo de DTD con declaración de atributos para el tipo de documento ejemplificado en la Figura 2.2.

```
<!ELEMENT Curso ( (asignatura | Profesor )+)>
<!ELEMENT asignatura ( nombre, area, cuatrimestre )>
  <!ATTLIST asignatura
            carácter ( obligatoria | optativa ) #REQUIRED >
<!ELEMENT Profesor ( nombre, Departamento, Facultad )>
  <!ATTLIST Profesor
             tipo CDATA #REQUIRED >
<!ELEMENT nombre ( #PCDATA )>
<!ELEMENT area ( #PCDATA )>
<!ELEMENT cuatrimestre ( #PCDATA )>
<!ELEMENT Departamento ( #PCDATA )>
<!ELEMENT Facultad ( #PCDATA )>
```

Figura 2.5. Ejemplo de DTD con declaración de atributos.

Mientras que las DTDs han resultado un mecanismo útil y sencillo para describir gramáticas documentales (véase, por ejemplo (Megginson et al, 1998)), durante su uso práctico se han detectado también diversas limitaciones, entre las que cabe destacar las siguientes:

–*Poder expresivo limitado*. Las facilidades para establecer restricciones sobre tipos de atributos y/o contenidos son limitadas (Birceck et al., 2001). Para ciertas aplicaciones, si se pudieran tipar los atributos y los contenidos, entonces no sería necesario tener que realizar comprobaciones explicitas, pues éstas se realizarían a nivel de validación del documento.

–*Falta de modularidad*. Con los mecanismos de modularización proporcionados por las DTDs, basados en *entidades de tipo parámetro* (Megginson et al, 1998), es complicado combinar DTDs para dar lugar a otras más complejas, a fin de extender DTDs preexistentes con nuevas estructuras. Así mismo, las DTDs no proporcionan soporte para definir cómo usar los espacios de nombres.

–*Dificultad en representar el desorden*. En algunas ocasiones puede ser interesante que los sub-elementos de un elemento no tengan un orden

[3] Cada elemento solo puede tener declarado un atributo del tipo `ID`. El valor que toma un atributo `ID` de un elemento no puede aparecer en ningún otro elemento del mismo documento.

definido. Sin embargo, con los mecanismos que ofrecen las DTDs resulta complicado especificar que estos puedan aparecer desordenados.

Como consecuencia de las limitaciones presentes en las DTDs han surgido alternativas para describir las gramáticas, extendiendo o complementando las capacidades expresivas de las DTDs (Lee et al., 2000), (Vlist, 2001), (Jelliffe, 2001), (Murata et al., 2001) (Murata et al., 2005). Entre estas alternativas destacan las siguientes:

- XML Schema (Fallside, 2001), (Thompson et al., 2001), (Biron et al., 2001), una especificación recomendada por el W3C, que permite describir un lenguaje de marcas formulando el tipo al que se ajusta la estructura de sus documentos. Para ello el lenguaje define varios tipos predefinidos (*string*, *integer*, *decimal*, *date* y *boolean*), y permite definir nuevos tipos como tipos más simples con restricciones añadidas o bien tipos complejos, así como utilizar todos ellos para caracterizar la estructura de los elementos documentales. Las definiciones de tipos se especifican en sintaxis XML usando etiquetas definidas en el propio lenguaje.
- RELAX NG (Clark et al., 2001a), (Clark et al., 2001b), una mezcla de los lenguajes de esquema RELAX (Murata, 2000) de orientación gramatical, y de TREX (Clark, 2001a), (Clark, 2001b) orientado a tipos. RELAX se basa en la teoría de lenguajes de árboles/bosques regulares (Thatcher, 1967), (Takahashi, 1975), (Thatcher, 1973), (Murata, 1995), (Comon et al., 1997), (Murata, 1999), y busca la facilidad de uso. Por su parte, TREX consiste en aplicar las expresiones regulares sobre árboles a la descripción de esquemas.
- Schematron (Jelliffe, 2002) es un lenguaje de esquema que modela el lenguaje de marcas definiendo restricciones sobre las estructuras que pueden y no pueden aparecer en el interior de un elemento, y se fundamenta en las llamadas *gramáticas de asertos* (Raggett, 1999).
- Examplotron (Vlist, 2003b) es un lenguaje de esquema que se basa en la especificación de ejemplos anotados con información gramatical y restricciones que permite la generalización de dichas anotaciones.

2.2 Tecnologías de Procesamiento de Documentos XML

Cuando se marca un documento con XML se hace explícita la estructura lógica del documento, pero no se dice nada acerca de cómo se debe procesar, dado que una característica esencial del marcado XML es ser *descriptivo*, es decir, separar la forma de estructurar los documentos de la forma en que deben ser procesados (Coombs et al., 1987), (Fuchs, 1997) . En este sentido la especificación no indica mecanismos de cómo llevar a cabo el procesamiento de documentos XML. Con esta última finalidad se han desarrollado distintas tecnologías de procesamiento de documentos XML, que pueden clasificarse en dos tipos de acuerdo a la especificidad de la tarea de procesamiento que se debe realizar: *tecnologías específicas* y *tecnologías de propósito general*. Los siguientes puntos desarrollan estos aspectos.

2.2.1 Tecnologías Específicas

Las tecnologías de procesamiento específicas abordan la resolución de determinados tipos de tareas específicas de procesamiento. Estas tareas se refieren principalmente a la consulta de información contenida en un documento o a la transformación de un documento de acuerdo a un esquema a otros esquemas diferentes.

Las tecnologías más extendidas para llevar a cabo este tipo de procesamientos se caracterizan por modelar los documentos XML como árboles cuyos nodos se corresponden con los elementos y atributos del documento. Así:

- Cada nodo de un elemento puede tener nodos hijos, que se corresponden con subelementos o atributos del elemento, y cada nodo de un elemento o de un atributo tiene un nodo padre que se corresponde con un elemento[4].
- Para modelar el orden de los elementos y de los atributos en el documento, se considera el orden de los nodos hijos en el árbol, y para modelar el contenido textual de un elemento, éste se considera como un nodo de texto, hijo del elemento correspondiente[5].

Actualmente destacan tres tecnologías de procesamiento específico:

```
/Curso/asignatura[nombre="Estructuras de datos"]
```

Figura 2.6. Ejemplo de expresión XPath.

- *XPath* (Clark et al., 1999), un lenguaje de consulta sobre árboles XML basado en describir expresiones regulares sobre rutas de acceso en los árboles documentales (Abiteboul et al. ,2000). Cada expresión de ruta consiste en un conjunto de pasos de ubicación separados por el símbolo "/", cuya evaluación de izquierda a derecha da como resultado un conjunto ordenado de nodos del documento denominado *conjunto de elementos actual*. Al inicio, el conjunto de elementos actual sólo contiene al nodo de la raíz del documento, que se representa como "/" en la expresión de ruta. A continuación, cuando en la expresión de ruta aparece el nombre de un elemento, entonces la evaluación da lugar a un conjunto ordenado de elementos actual formado por los hijos de los elementos del conjunto de elementos actual anterior[6]. De manera similar es posible acceder a los valores de los atributos de los elementos usando el símbolo @. Las condiciones de selección se especifican usando predicados situados junto a cualquier paso en una ruta, encerrados entre corchetes, y funciones que se usan como parte de los predicados. También es posible el uso de determinados operadores a nivel de paso como por ejemplo "//" para saltar varios niveles de nodos, o a nivel de expresión de ruta como "|" para unir los resultados de dos expresiones diferentes. En la Figura 2.6 se muestra un ejemplo de expresión XPath, en la que se devuelven todos los elementos `asignatura` cuyo nombre sea "`Estructura de datos`".

[4] Salvo el elemento raíz.

[5] A veces los elementos contienen texto descompuesto en subelementos, en cuyo caso se consideran que están formados por varios nodos de texto hijo.

[6] Los nodos que se devuelven en cada evaluación siempre aparecen en el mismo orden que en el documento.

(a)

```
<Curso>
   <asignatura>
           <nombre> Estructuras de datos </nombre>
           <area> Software </area>
           <cuatrimestre> Primero </cuatrimestre>
   </asignatura>
   <asignatura>
           <nombre> Lógica Matemática </nombre>
           <area> Ciencias de la computación </area>
           <cuatrimestre> Anual </cuatrimestre>
   </asignatura>
</Curso>
```

(b)

```
<Asignaturas>
     for $x in
        /Curso/asignatura/nombre
     return
          <nombre>$x/text() </nombre>
</Asignaturas>
```

(c)

```
<Asignaturas>
          <nombre> Estructura de datos </nombre>
          <nombre> Lógica Matemática </nombre>
</Asignaturas>
```

Figura 2.7. (a) Ejemplo de documento XML,(b) Ejemplo de consulta XQuery, (c) Resultado de aplicar b) al documento a).

– *XQuery* (Draper et al. 2003), un lenguaje de consultas normalizado de XML propuesto por el consorcio W3C. Las consultas se basan en expresiones FLWOR (*For Let Where Order-by Return*) que pueden contener: (i) cláusulas *for* (proporcionan variables que contienen el resultado de evaluar una expresión XPath; si se especifican varias variables entonces se trata como el producto cartesiano de los valores posibles que las variables pueden tomar), (ii) cláusulas *let* (asignan el resultado de la evaluación de una expresión XPath a una variable dada), (iii) cláusulas *where* (permiten realizar comprobaciones sobre las tuplas devueltas por una cláusula *for*), (iv) cláusulas *order by* (permiten la ordenación de los resultados de una consulta), y (v) cláusulas *return* (permiten construir resultados en XML). Estas expresiones extienden a las expresiones XPath, añadiendo capacidades de iteración sobre colecciones de nodos, procesamiento condicionado, y transformación y construcción de estructuras documentales. Una característica muy interesante de XQuery es que se trata de un lenguaje tipado estáticamente, que usa el sistema de tipos de XML Schema. En este sentido el lenguaje proporciona conversión automática de tipos cuando sea necesario, el uso de funciones de conversión de tipos, y otras características tales como la especificación parcial de tipos o la referencia a una secuencia de valores de un tipo mediante la adición de un nombre de tipo al operador "*". En la Figura 2.7 se muestra, en b), una consulta XQuery que, aplicada sobre el documento que aparece en a), genera como resultado el documento de c).

(a)

```
<Curso>
   <asignatura>
           <nombre> Teoría de Autómatas y Lenguajes formales </nombre>
           <area> Ciencias de la computación </area>
           <cuatrimestre> Anual </cuatrimestre>
   </asignatura>
   <Profesor>
           <nombre> Peter Strauss </nombre>
           <Departamento> Ciencias de la Computación </Departamento>
           <Facultad> Informática </Facultad>
   </Profesor>
</Curso>
```

(b)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:output
   method="xml" indent="yes" encoding="iso-8859-1"/>
<xsl:template match="Curso">
<html>
<head>
<xsl:apply-templates select="Asignatura" mode="head"/>
</head>
<body>
<xsl:apply-templates select="Asignatura" mode="body"/>
<xsl:apply-templates select="Profesor"/>
</body>
</html>
</xsl:template>
<xsl:template match="Asignatura" mode="head">
<title><xsl:value-of select="area"/></title>
</xsl:template>
<xsl:template match="Asignatura" mode="body">
<h1><xsl:value-of select="nombre"/></h1>
<h2><xsl:value-of select="Cuatrimestre"/></h2>
</xsl:template>
<xsl:template match="Profesor">
<ol>
<xsl:apply-templates select="*"/>
</ol>
</xsl:template>
<xsl:template match="nombre">
<li><xsl:value-of select="."/></li>
</xsl:template>
</xsl:stylesheet>
```

(c)

```
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
 <head>
  <title>Ciencias de la Computación</title>
 </head>
 <body>
  <h1> Teoría de Autómatas y Lenguajes formales </h1>
  <h2> Anual </h2>
  <ol>
   <li> Peter Strauss </li>
  </ol>
 </body>
</html>
```

Figura 2.8. (a) Ejemplo de documento XML, (b) Ejemplo de transformación XSLT y (c) Resultado de aplicar b) al documento a).

– *XSLT* (Clark ,1999), (Kay, 2007) es un lenguaje recomendado por la W3C para la realización de transformaciones sobre documentos XML. Las transformaciones se expresan mediante reglas recursivas denominadas *plantillas***.** Las plantillas permiten seleccionar contenido, generar fragmentos del documento resultante de la transformación y aplicar plantillas a modo de funciones sobre subárboles del documento transformado (esta característica se denomina *recursión estructural*). La selección de contenido se lleva a cabo utilizando patrones XPath. Por su parte, la generación de fragmentos de documento se lleva a cabo seleccionado y extrayendo contenido del

documento original, indicando nuevo contenido literal, aplicando la característica de recursión estructural, así como utilizando otras características más avanzadas del lenguaje (e.g., facilidades para ordenación de nodos, uso funcional de plantillas,…). Por último mencionar que aunque XSLT no es un lenguaje tipado, se han hecho algunas propuestas de tipado estático sobre subconjuntos del lenguaje basadas en la teoría de lenguajes regulares sobre árboles (véase, por ejemplo (Tozawa, 2001)). En el ejemplo de la Figura 2.8 se muestra en b) una transformación XSLT que aplicada sobre el documento que aparece en a), genera como resultado el documento de c).

Además de las tecnologías descritas, existen otras tecnologías menos extendidas al procesamiento de XML, como el nivel 2 de las CSS (*Cascading Style Sheets*) (Bos et al., 1998), que permite la realización de ciertos tipos de transformación, o XSL (Adler et al., 2001), refinamiento de XSLT donde los documentos que resultan en la transformación son descripciones de árboles de objetos de presentación.

2.2.2 Tecnologías de Propósito General

Cuando las tareas de procesamiento que se precisan realizar sobre los documentos XML son demasiado específicas o complejas, y no existen tecnologías de procesamiento específicas adecuadas para llevarlas a cabo, se recurre a tecnologías de propósito general. Estás tecnologías se presentan como marcos de procesamiento embebidos en un lenguaje de programación de propósito general (Java, C++,…), dando lugar a aplicaciones cuya estructura habitual se refleja en la Figura 2.9.

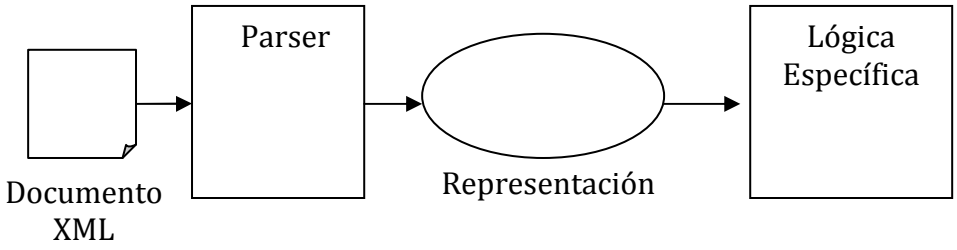


Figura 2.9. Estructura de las aplicaciones de los marcos de procesamiento de XML de propósito general.

De acuerdo a la estructura esbozada en la Figura 2.9, los marcos genéricos constan de un analizador o *parser* XML que se encarga de leer el documento XML, comprobar su validez mediante un análisis léxico y sintáctico, y ofrecer una *representación* de los documentos procesados junto con una interfaz para manipular y consultar los elementos de información contenidos en los documentos (Birceck et al., 2001), (Lam et al., 2008). Utilizando el lenguaje de propósito general del marco se programa el procesamiento que se desea realizar sobre los documentos, obteniendo como resultado la *lógica específica de la aplicación*. Esta capa de software accede a la información de los documentos a través de los métodos de la interfaz ofrecida por el *parser*[7].

[7] El marco normaliza la conexión entre el *parser* y la lógica específica a través de la representación intermedia de los documentos procesados.

A nivel tecnológico, con respecto a los *parsers* XML contenidos en los marcos, existen propuestas como JAXP (Mordani et al., 2002) que define un marco orientado a Java para estandarizar el uso de *parsers* XML en procesadores de aplicaciones XML intentando independizar el procesador del *parser* concreto utilizado, o bien aproximaciones más generalistas (Laurent et al., 1999), (Ahmed et al. ,2001), (Birceck et al., 2001) que definen marcos genéricos de procesamiento que incluyen distintas representaciones para los documentos XML procesados, entre las que destaca la adoptada por *Xerces*. Este último es un *parser* desarrollado en el proyecto XML *Apache* que ofrece gran modularidad y capacidad de configuración, y que implementa un modelo de pipelines (Prud'hommeaux., 2001) que une los componentes de análisis y validación[8], de manera que tomando como entrada los documentos XML, produce como salida una representación adecuada de los mismos, tal como se muestra en la Figura 2.10.

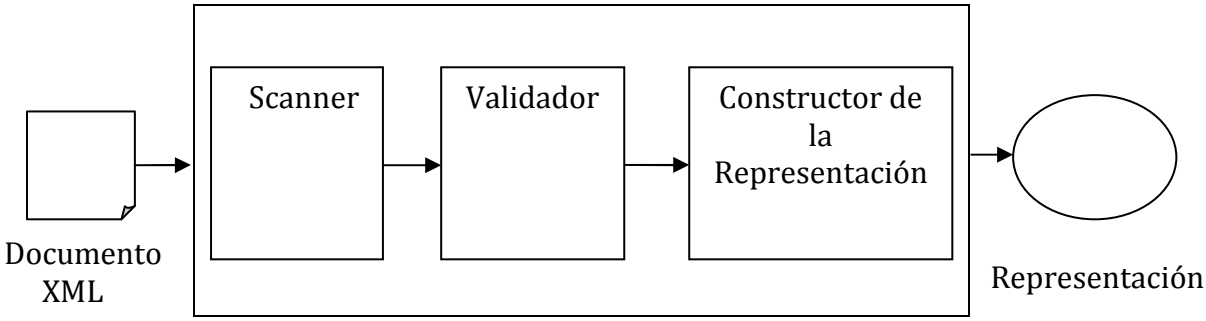


Figura 2.10. Estructura de un parser como pipeline de componentes.

Las tecnologías concretas que implementan los marcos de procesamiento genéricos se pueden clasificar de acuerdo a la naturaleza de la interfaz a los documentos que proporciona el *parser*. De esta forma, es posible distinguir entre:

- *Marcos orientados a árboles*. Se basan en el hecho de que la estructura lógica de un documento XML corresponde a un árbol, de manera que este tipo de interfaces ofrecen operaciones para recorrer el árbol.
- *Marcos orientados a eventos***.** Representan los documentos como una secuencia ordenada de los eventos[9] que suceden al recorrer el árbol del documento en un orden[10] determinado, de manera que la lógica específica consiste en acciones para tratar los eventos que van surgiendo.

A continuación se analizan con más detalle las características de cada uno de estos tipos de marco, a través de sus ejemplos más característicos.

2.2.2.1 Marcos orientados a árboles: DOM y marcos de vinculación de datos

Los marcos de procesamiento orientados a árboles utilizan, como ya se ha indicado, una representación explícita del documento procesado como un árbol. De esta forma, su principal ventaja es que permiten recorrer el documento durante su

[8] Cada disposición distinta de componentes en el pipeline se denomina *configuración*, existiendo un marco denominado XNI que permite añadir nuevos componentes y nuevas configuraciones, tales como distintos tipos de validadores o cambios en la sintaxis concreta de los documentos. La comunicación entre los componentes del pipeline se realiza mediante serializaciones de los *infosets* (Cowan et al., 2001) de los documentos XML analizados.

[9] Un evento consiste en encontrarse con un elemento del documento o con determinada información relevante.

[10] En general el orden de recorrido es de izquierda a derecha y en profundidad

procesamiento de cualquier forma, sin limitarse a un recorrido prefijado. Así mismo, permiten realizar actualizaciones del documento eficientemente. Sin embargo presentan como desventaja, el hecho de que, para procesar el documento, se necesita disponer de la representación explícita del árbol documental, lo cual es normalmente ineficiente para documentos grandes, o documentos que se construyen incrementalmente de manera asíncrona (Lam et al., 2008).

A continuación se analizan un par de ejemplos representativos de enfoques de procesamiento orientados a árboles: la especificación DOM, y las propuestas de vinculación de datos.

2.2.2.1.1 DOM

DOM (Document Object Model) (DOM, 2000) es una especificación de la W3C que define las interfaces que permiten manipular el contenido de un documento XML a partir de su representación lógica en forma de árbol. Para ello en DOM se establece un modelo estructural orientado a objetos para árboles de documentos XML independiente de la implementación[11], en el que cada elemento de un documento se representa como un nodo de un árbol. La especificación define una clasificación de nodos que refleja los tipos de elementos estructurales de XML. Así se distingue entre nodos de tipo *elemento*, caracterizados por tener asociada una secuencia de nodos hijos y una tabla de atributos cuyos valores son cadenas de caracteres, y nodos de tipo *texto*, que toman como valor un contenido textual. Entre los nodos elemento se establecen relaciones con su primer y último hijo, y entre nodos hermanos consecutivos. De esta forma, para recorrer el árbol se comienza por su raíz y se itera sobre la secuencia de los nodos hijos de los elementos o se utilizan las relaciones entre nodos.

Desde un punto de vista estructural, el modelo de DOM representa las características básicas de la estructura de un documento XML bien formado[12]. Sin embargo en determinados contextos puede ser necesario ampliar las características representadas de un documento XML. Para ello DOM define un conjunto de niveles de acuerdo a las características que capturan, y que permite la extensión de DOM. Así, por ejemplo, el *nivel 1* (Apparao et al., 1998) está formado por el modelo estructural básico. Por su parte, el *nivel 2* (Hors et al., 2000a), (Hors et al., 2000b), (Kesselman et al., 2000), (Pixle, 2000), (Stenback et al., 2003), (Wilson et al., 2000) añade al modelo básico características como el espacio de nombres, o la propagación de eventos a través de los árboles. Por último, el *nivel 3* (Chang, 2003), (Hégaret, 2003), (Hors et al., 2003), (Stenback, 2003), (Whitmer, 2003) está dedicado a las relaciones que existen entre documentos y árboles documentales.

A fin de ejemplificar el uso de DOM, en la Figura 2.11 se introduce una DTD para un sub-lenguaje del lenguaje de marcado para el ejemplo de las asignaturas de un curso utilizado anteriormente. La Figura 2.12 muestra un ejemplo de lógica específica programada en Java para un procesador basado en DOM para la DTD de la Figura 2.11, el cual permite construir una representación en forma de lista de los

[11] Es decir, en términos de *interfaces* que son posteriormente implementadas por cada implementación.
[12] Este conjunto de propiedades se recoge en la especificación XML Infoset (Cowan et al., 2001).

contenidos de un documento XML de un curso. Para ello se dispone del conjunto de operaciones que se comenta a continuación:

- El método `construyeCurso` obtiene una representación del curso contenida en el documento asociado con dicho objeto.
- El método `esAsignatura` determina si un nodo se corresponde con un elemento de tipo `Asignatura`.
- Los métodos `nombre` y `area` obtienen el nombre y área de una asignatura.
- El método `elementoHijo` accede al elemento hijo de un nodo por posición.
- El método `contenido` recupera el contenido textual de un nodo, recorriendo en preorden el subárbol del nodo y acumulando el contenido mediante el método `recolectaContenido`.

Las características de DOM usadas corresponden al nivel 1. Se han utilizado los métodos de navegación sobre el árbol DOM de la interfaz `Node`: `getChildNodes` (nodos hijo de uno dado), `getFirstChild` (primer hijo de un nodo dado) y `getNextSibling` (nodo hermano que sucede a uno dado en la secuencia ordenada de hijos del nodo padre). Así mismo, se han usado los métodos `getLength` (longitud de la lista) e `Item` (indexa los elementos de un lista de nodos) de `NodeList` para poder iterar sobre la lista de nodos hijo.

```
<!ELEMENT curso ( (asignatura)+)>
<!ELEMENT asignatura ( nombre, área?)>
<!ELEMENT nombre ( #PCDATA )>
<!ELEMENT area ( #PCDATA )>
```

Figura 2.11. Ejemplo de DTD para el tipo de documento sobre asignaturas de un curso.

DOM comparte todas las ventajas inherentes a los marcos de procesamiento orientados a árboles. Así mismo, dado que DOM se especifica en términos de interfaces (en el sentido de lenguajes orientados a objetos como Java), es posible cambiar la implementación de manera transparente, sin necesidad de modificar la lógica específica de la aplicación. Por otra parte, a nivel de implementación es posible utilizar técnicas que mitiguen el coste en memoria del mantenimiento explícito del árbol documental (e.g., implementaciones *perezosas*, que construyen el árbol documental conforme éste se precisa, así como implementaciones que mantienen el árbol en una base de datos externas, manteniendo en memoria únicamente una porción del mismo). No obstante, la principal desventaja inherente a DOM es su genericidad. Efectivamente, DOM trata de representar cualquier árbol XML con independencia de la aplicación específica subyacente. Para ello se representa mediante el modelo de objetos los conceptos genéricos (elementos, atributos,….) que aparecen reflejados en los documentos marcados, obteniendo una representación genérica (modelo documental de datos). De esta manera, la lógica específica del procesador debe realizar habitualmente un proceso previo de traducción, denominado *vinculación de datos* en el que se traduce la representación genérica en una representación específica en términos de los

conceptos documentados (modelo de datos dependiente del lenguaje de marcado concreto)[13] tal como se sugiere en la Figura 2.13.

```
class Asignatura {
  private String nombre;
  private String area;
  public Asignatura(String nombre,String area) {
   this.nombre = nombre;
   this.area = area;
  }
  public String nombre() {return nombre;}
  public String area() {return area;}
}

class ProcesadorCurso {
  public static List construyeCurso(Document doc) {
     List curso = new LinkedList();
     Element infoCurso = doc.getDocumentElement();
     NodeList infoAsignaturas = infoCurso.getChildNodes();
     for (int i=0; i < infoAsignaturas.getLength(); i++)
     if (esAsignatura(infoAsignaturas.item(i))) {
       String nombre = nombre(infoAsignaturas.item(i));
       String area = area(infoAsignaturas.item(i));
       curso.add(new Asignatura(nombre,area));
     }
     return curso;
    }
  private static boolean esAsignatura(Node nodo) {
   return nodo.getNodeType() == Node.ELEMENT_NODE &&
         ((Element)nodo).getTagName().equals("Asignatura");
  }
  private static String nombre(Node infoAsignatura) {
     Node nombre = elementoHijo(infoAsignatura,0);
     return contenido(nombre);
  }
  private static String area(Node infoAsignatura) {
    Node area = elementoHijo(infoAsignatura,1);
    if (area == null) return "";
      else return contenido(area);
  }
  private static Node elementoHijo(Node nodo, int num) {
    Node hijo = nodo.getFirstChild();
    while (hijo != null) {
      if (hijo.getNodeType() == Node.ELEMENT_NODE) num--;
      if (num < 0) break;
      hijo = hijo.getNextSibling();
     }
    return hijo;
  }
  private static String contenido(Node nodo) {
    StringBuffer contenido = new StringBuffer();
    recolectaContenido(nodo,contenido);
    return contenido.toString();
  }
  private static void recolectaContenido(Node nodo,
                                         StringBuffer contenido) {
    if (nodo.getNodeType() == Node.TEXT_NODE ||
    nodo.getNodeType() == Node.CDATA_SECTION_NODE)
    contenido.append(nodo.getNodeValue());
    NodeList hijos = nodo.getChildNodes();
    for (int i=0; i < hijos.getLength(); i++)
    recolectaContenido(hijos.item(i),contenido);
  }}
```

Figura 2.12.Ejemplo de Lógica específica basada en DOM para un procesador de construcción de cursos.

[13] A partir del resultado de la vinculación de datos se realiza lo que resta del procesamiento, el cual se implementa en términos de la representación específica.

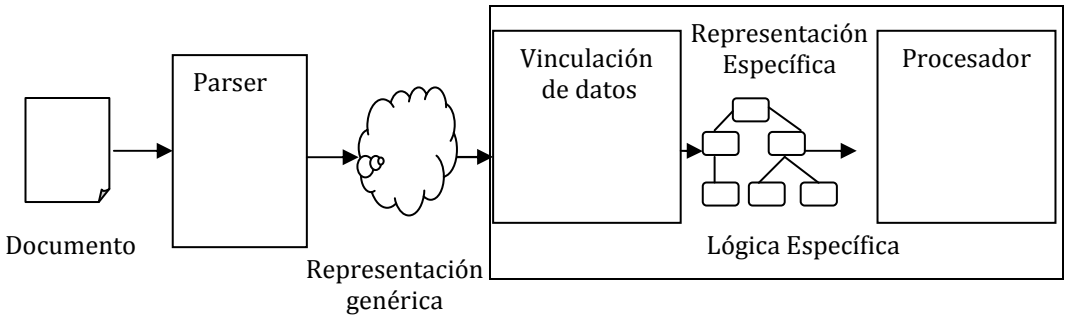


Figura 2.13. Esquema de la Lógica específica de un procesador.

2.2.2.1.2 Marcos de Vinculación de Datos

Los marcos de vinculación de datos están orientados a evitar la programación explícita de la operación de vinculación. Efectivamente, esta operación sería innecesaria si se generase directamente la representación específica. Con el objetivo de eliminar este procesamiento adicional, se han desarrollado las denominadas propuestas de *vinculación de datos* (Birceck et al., 2001), que propugnan la generación de interfaces orientadas a árboles específicas para cada aplicación concreta.

Las propuestas de vinculación de datos se basan en la idea de utilizar marcos de alto nivel para realizar la vinculación de datos (Bourret, 2003), en vez de implementarla manualmente utilizando un lenguaje de programación de propósito general. Los marcos se caracterizan porque toman como entrada la gramática documental de una aplicación XML, y generan como resultado un componente que, conectado al *parser*, proporciona representaciones de los árboles documentales en términos específicos de la aplicación. En general, los marcos son bidireccionales, dado que permiten la traducción desde documentos o representaciones genéricas a representaciones específicas (*unmarshaling*), y desde las representaciones específicas a documentos (*marshaling*). Un ejemplo de marco de vinculación de datos es *Castor*, un marco que permite la vinculación de datos con el modelo de objetos de JavaBeans (JavaBeans, 1997). Este marco toma como entrada un esquema de definición de un lenguaje de marcas y un documento marcado de asociación[14] que actúa a modo de especificación declarativa de cómo realizar la vinculación, y genera como resultado una representación específica para el lenguaje de marcas en forma de modelo de objetos.

Aunque las propuestas de vinculación de datos reducen la complejidad del procesamiento que debe realizar la lógica de procesamiento, presentan como desventaja principal la inflexibilidad del proceso de generación que, aun siendo modulable, no siempre permite obtener el tipo de interfaz deseado (funciona bien con documentos muy estructurados, pero no así con documentos en los se mezclan contenidos textuales y marcado). Además comparten con el resto de los enfoques orientados a árboles la dificultad para manejar documentos muy grandes o generados incrementalmente de forma asíncrona sobre un flujo (*stream*) de información. Así mismo, debido al carácter específico de las representaciones generadas, la realización de las optimizaciones facilitadas por la separación entre

[14] Por defecto la asociación entre documentos y objetos se realiza utilizando los mecanismos de introspección de JavaBeans.

interfaz e implementación en DOM no son, en este caso, conseguibles de forma tan directa y general como en aquel marco.

2.2.2.2 Marcos orientados a eventos

Los marcos orientados a eventos intercalan el procesamiento durante el recorrido del documento, de forma similar a los traductores en una pasada utilizados en compilación de lenguajes de programación (Aho et al., 2006). De esta forma, respecto a las representaciones orientadas a árboles, presentan la ventaja de requerir menos memoria que éstas, dado que no se necesita disponer explícitamente del árbol del documento, ni del documento completo para comenzar a procesarlo. Así mismo, dado que el procesamiento se intercala con el recorrido del documento, estos marcos posibilitan el procesamiento asíncrono de documentos, lo que los hace idóneos para el procesamiento de flujos de información XML. Sin embargo presentan como desventajas la necesidad de tener que acoplar el procesamiento con el recorrido del documento en un sentido determinado, así como de forzar un procesamiento secuencial de dicho documento, lo que implica un mayor coste en las operaciones que involucran la modificación del documento (Lam et al., 2008).

Los marcos orientados a eventos siguen habitualmente uno de los siguientes enfoques:

- Enfoques *push*. Los enfoques *push* se caracterizan porque el control reside en el *parser* XML. La lógica específica se conecta al *parser* registrando en el mismo un conjunto apropiado de *manejadores de eventos*, que el parser invoca cada vez que descubre el correspondiente evento significativo en el documento. La principal ventaja de estos enfoques es que permiten aprovechar la extensibilidad inherente al modelo de cómputo orientado a eventos (Dabek et al., 2002). Efectivamente, los procesadores pueden concebirse como capas de tratamientos de eventos que, a su vez, pueden propagar los eventos a capas superiores. Su principal desventaja, sin embargo, reside en la necesidad de mantener el contexto de procesamiento entre las invocaciones de tratamiento de eventos debido a que existe un desacoplamiento entre las acciones de tratamiento de eventos que se llevan a cabo y el proceso de análisis.
- Enfoques *pull*. En estos enfoques, al contrario que en los *push*, el control reside en la lógica específica, que invoca al *parser* conforme requiere un nuevo evento significativo. En este sentido, el *parser* XML juega un papel similar a un *analizador léxico* en un traductor convencional, y las aplicaciones pueden construirse siguiendo estrategias análogas a las de los traductores predictivos recursivos (Aho et al., 2006). De esta forma, el problema de mantenimiento de contexto desaparece, ya que éste puede mantenerse en el espacio de control de la lógica específica (normalmente organizada como un conjunto de operaciones mutualmente recursivas). Sin embargo, este tipo de enfoques conducen a aplicaciones más difíciles de extender incrementalmente.

A continuación se analiza un ejemplo significativo de cada uno de los enfoques: SAX en relación con los enfoques *push*, y StAX en relación con los enfoques *pull*.

2.2.2.2.1 SAX

SAX (*Simple Api for XML)* es el estándar de hecho para los enfoques *push* (Brownell, 2002). Siguiendo la filosofía *push*, las aplicaciones de procesamiento SAX constan de un *parser* básico conforme con SAX, que es extendido mediante el registro de un conjunto de manejadores de eventos (objetos en los que se delega el tratamiento de los eventos). De esta manera, cuando un analizador SAX procesa un documento y ocurre un evento, entonces se invoca con los parámetros que lo describen a la operación controladora adecuada para ese evento en el manejador correspondiente situado en la extensión, operación que lleva a cabo su función.

Los principales componentes que componen SAX son la interfaz `XMLReader`, que especifica el contrato que deben cumplir los procesadores básicos, y las interfaces `ContentHandler`, `ErrorHandler`, `EntityResolver` y `DTDHandler`, que caracterizan los distintos manejadores de eventos. Así mismo, SAX introduce la interfaz `XMLFilter`, que extiende a la interfaz `XMLReader` para facilitar la construcción de procesadores SAX a partir de otro procesador SAX. En este sentido cabe mencionar que los procesadores SAX pueden construirse por capas de extensión, definiendo una arquitectura modular multinivel, de manera que el procesador de la capa de nivel *n* tiene como padre al procesador de la capa de nivel *n-1*, y es el padre del procesador de la capa de nivel *n+1*. Cada procesador trata los eventos producidos por su procesador padre y notifica eventos a su extensión en el nivel inmediatamente superior. Además los procesadores delegan en sus procesadores padres determinadas actividades, como, por ejemplo, el análisis del documento.

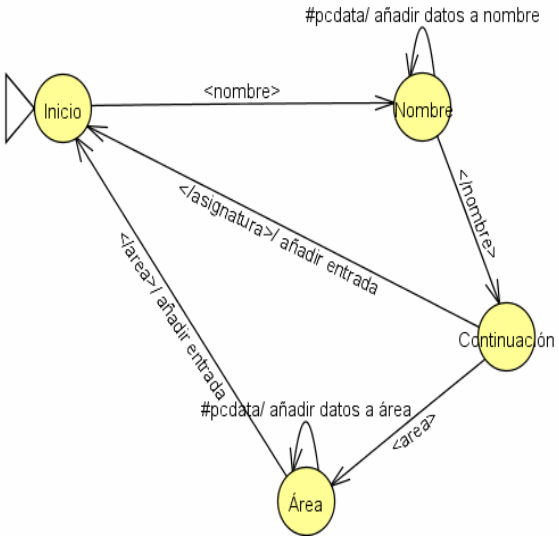


Figura 2.14 Diagrama de estados para el procesador SAX.

A fin de ejemplificar el uso de SAX, se va a considerar nuevamente la DTD de la Figura 2.11, que representaba un sub-lenguaje de marcado para el ejemplo de las asignaturas de un curso, y se va a construir la lógica específica de un procesador

SAX en Java que construye la representación en forma de lista de los contenidos de un curso. En general, para llevar a cabo un diseño sistemático puede realizarse un diagrama de estados que especifica los posibles eventos, cómo responder a ellos, y la forma de mantener el contexto de procesamiento (Ahmed et al. ,2001). En la Figura 2.14 se muestra el diagrama de estados que define la lógica específica del procesador del ejemplo. De esta forma, se tienen los siguientes estados:

```
class ProcesadorCurso {
   public void trataApertura(Procesador procesador,String etiqueta) {}
   public void trataCierre(Procesador procesador,String etiqueta) {}
   public void trataDatos(Procesador procesador, char ch[],int inicio,
                                int longitud) {}
   }

class Inicio extends ProcesadorCurso{
   private static Inicio inicio = null;
   public static Inicio estado() {
        if (inicio == null) inicio = new Inicio();
        return inicio;
   }
   public void trataApertura(ProcesadorCatalogo procesador,String etiqueta) {
       if (etiqueta.equals("nombre")) { procesador.iniciaDatos();
       procesador.fijaEstado(Nombre.estado());
   }}}

class Nombre extends ProcesadorCurso {
   private static Nombre nombre = null;
   public static Nombre estado() {
      if (nombre == null) nombre = new Nombre();
      return nombre;
   }
   public void trataCierre(Procesador procesador, String etiqueta) {
      if (etiqueta.equals("nombre"))
                        procesador.fijaEstado(Continuacion.estado());
   }
   public void trataDatos(Procesador procesador,char[] datos, int comienzo,
                             int longitud) {
      procesador.concatenaANombre(datos,comienzo,longitud);
   }}

class Area extends Procesador Curso{
   private static Area area = null;
   public static Area estado() {
      if (area == null) area = new Area();
      return area;}
   public void trataCierre(Procesador procesador, String etiqueta) {
      if (etiqueta.equals("area")) { procesador.insertaEntrada();
      procesador.fijaEstado(Inicio.estado());
   }}
   public void trataDatos(Procesador procesador,char[ ] datos, int comienzo,
        int longitud) {
   procesador.concatenaAArea(datos,comienzo,longitud);
   }}

class Continuacion extends ProcesadorCurso {
   private static Continuacion continuacion = null;
   public static Continuacion estado() {
       if (continuacion == null) continuacion = new Continuacion();
       return continuacion; }
   public void trataApertura(Procesador procesador,String etiqueta) {
       if (etiqueta.equals("area")) procesador.fijaEstado(Area.estado());
   }}
   public void trataCierre(Procesador procesador,String etiqueta) {
       if (etiqueta.equals("asignatura")) { procesador.insertaEntrada();
       procesador.fijaEstado(Inicio.estado());
   }}}
```

Figura 2.15. Estados para la lógica específica del procesador SAX.

– *Inicio*. Representa el estado inicial del procesador, en el que se espera información acerca de una asignatura, y se asocia con el evento de la apertura

de una etiqueta `nombre`. Cada vez que se regresa al estado inicial se crea una nueva entrada en la lista de asignaturas.

– *Nombre*[15].En este estado se recoge el texto del nombre de la asignatura. Se transita al siguiente estado cuando se encuentra la etiqueta de cierre `</nombre>`.

– *Continuación*. En este estado se decide si se ha finalizado el procesamiento de una asignatura (caso de aparecer una etiqueta de cierre `</asignatura>`), o bien si la asignatura tiene un área asociada (caso de aparecer una etiqueta de apertura de `<area>`).

– *Área.* En este estado se recoge el texto asociado con el área de la asignatura. Se transita al siguiente estado cuando se encuentra la etiqueta de cierre `</area>`.

```
class Procesador extends DefaultHandler {
   private List<Asignatura> curso;
   private StringBuffer nombre;
   private StringBuffer area;
   private ProcesadorCurso estado;
   public Procesador() {
      curso = new LinkedList<Asignatura>();
      nombre = new StringBuffer();
      area = new StringBuffer();
      fijaEstado(Inicio.estado());
   }
   public List curso() {return curso;}
// Manejo de eventos relativos a la estructura
   public void startElement(String uri,String nombreLocal,String etiqueta,
   Attributes atributos) {
       estado.trataApertura(this,etiqueta);
   }
   public void endElement(String uri,String nombreLocal,String etiqueta) {
       estado.trataCierre(this,etiqueta);
   }
   public void characters(char ch[], int inicio, int longitud) {
       estado.trataDatos(this,ch,inicio,longitud);
   }
// Manejo de errores
   public void warning(SAXParseException e) throws SAXException
                                          {trataError(e);}
   public void error(SAXParseException e) throws SAXException {trataError(e);}
   public void fatalError(SAXParseException e) throws SAXException
                                          {trataError(e);}
   private void trataError(SAXParseException e) throws SAXException {
       System.err.println("Error ("+e.getLineNumber()+"
                          ,"+e.getColumnNumber()+"):"+e.getMessage());
       throw e;
   }
// Manipulación del contexto
   public void fijaEstado(ProcesadorCurso estado) {this.estado = estado;}
   public void iniciaDatos() {
       nombre.delete(0,nombre.length());
       area.delete(0,area.length());
   }
   public void insertaEntrada() {
       curso.add(new Asignatura(nombre.toString(),area.toString()));
   }
   public void concatenaANombre(char[] datos,int comienzo,int longitud) {
       nombre.append(datos,comienzo,longitud);
   }
   public void concatenaAArea(char[] datos,int comienzo,int longitud) {
       area.append(datos,comienzo,longitud);
   }
}
```

Figura 2.16. Manejador de eventos que captura la lógica específica del procesador SAX.

[15]Este estado igual que el estado Área tiene un bucle, dado que el procesador básico de SAX puede anunciar múltiples eventos para cada secuencia de contenido #PCDATA en el documento.

La Figura 2.15 muestra la implementación de los estados de la lógica específica del procesador mediante las clases `Inicio`, `Nombre`, `Continuacion` y `Area`. Cada una de las clases se ha definido como una especialización de la clase `ProcesadorCurso`. En la implementación se puede observar que ha sido necesario utilizar dos variables para almacenar el contexto: el nombre y el área de cada asignatura.

Por otra parte en la Figura 2.16 se muestra la implementación del manejador de los eventos que activa la lógica específica en función de los eventos definidos.

SAX, siendo el prototipo de los enfoques *push*, exhibe todas las ventajas de estos enfoques discutidas anteriormente. Así mismo, exhibe todas sus desventajas, y comparte también con DOM la desventaja inherente a la genericidad de la representación (la representación de los documentos como flujos de eventos SAX es a nivel del lenguaje XML, y no a nivel de cada una de sus aplicaciones).

2.2.2.2.2 StAX

StAX (*Streaming API for XML*) es una especificación representativa del estilo de procesamiento *pull* (McLaughlin, 2006). En este sentido, StAX se caracteriza por procesar XML como *streams* de *eventos*. Aunque SAX también permite manipular XML como flujos de eventos, el concepto de *evento* manejado en StAX difiere del manejado en SAX: StAX no necesita de un manejador que reciba los eventos del analizador según los va generando, y es la propia aplicación la que invoca al analizador y recupera los eventos uno tras otro. De esta forma, StAX no necesita mantener el estado del analizador y permite simplificar la codificación de la manipulación del contenido. Con respecto a DOM ofrece las mismas ventajas que SAX. Así, el procesamiento basado en eventos evita la necesidad de disponer de un modelo de objetos en memoria y totalmente construido para poder llevar a cabo el procesamiento, dado que en StAX, al igual que en SAX, simplemente se van tomando acciones en base a los eventos entrantes.

StAX *p*ermite dos estilos diferentes de análisis:

- El estilo definido en *Cursor API* facilita un procesamiento basado en un *cursor* que, en cada momento, se sitúa sobre un evento significativo del documento XML. Para hacer uso de este enfoque es necesario utilizar la interface `XMLStreamReader`, que proporciona métodos para desplazar el cursor sobre el flujo de eventos.
- Por su parte, el API *Event Iterator* ofrece un estilo alternativo, en el que se tiene acceso a los eventos particulares del documento, representados en forma de objetos. Para ello se usa la interface `XMLEventReader`, que permite ir visitando los eventos significativos en el documento, representados en forma de objetos de tipo `XMLEvent` (*interface* básica de los objetos de evento).

A fin de ejemplificar el uso de StAX, se va a considerar nuevamente el ejemplo de los documentos de descripción de curso, y se va a abordar su transformación a listas utilizando los dos estilos introducidos:

- La Figura 2.17 ilustra el procesamiento siguiendo el estilo *Cursor* API. En primer lugar se obtiene una instancia de `XMLInputFactory` y, a partir de ella, se crea una instancia de `XMLStreamReader` para analizar el documento. A continuación se recorre el flujo de eventos, y se extrae información de un

evento si éste es de tipo `START_ELEMENT`. Si el elemento recuperado es de tipo `nombre` o `area`, se almacena el contenido, respectivamente, en las variables `nombreAsignatura` y `areaAsignatura`. Una vez que se dispone de la información del nombre y área de cada asignatura, ésta se almacena en la lista de asignaturas. El proceso se repite hasta que se llega a un evento del tipo `END_DOCUMENT`, momento en el cual se cierra el analizador y la entrada.

```
import java.util.*;
import javax.xml.stream.*;
import java.io.*;

public class StAXXMLStreamReaderCurso {
    public List<Asignatura> procesaCurso(String curso) {
        FileReader      reader = null;
        XMLStreamReader parser = null;
        String  nombreAsignatura  = null;
        String  areaAsignatura    = null;
        int     event;
        String  tag;
        List<Asignatura> asignaturas = new LinkedList<Asignatura>();
        try {
            reader = new FileReader(curso);
            parser = XMLInputFactory.newInstance().createXMLStreamReader(reader);
            int event = parser.getEventType();
            while (true) {
                  if (event == XMLStreamConstants.START_ELEMENT){
                       tag = parser.getLocalName();
                       if ("nombre".equals(tag)){
                            nombreAsignatura = parser.getElementText();
                          } else if ("area".equals(tag)){
                            areaAsignatura = parser.getElementText();
                            asignaturas.append(new Asignatura(nombreAsignatura,
                                                              areaAsignatura);
                       }
                     }
                  if (!parser.hasNext()) break;
                  event = parser.next();
                }
            }

        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                reader.close();
            } catch (Exception ex){}
            try {
                parser.close();
            } catch (Exception ex){}
        }
       return asignaturas;
    }
}
```

Figura 2.17. Ejemplo de procesamiento StAX usando un estilo *Cursor* API.

– Por su parte, la Figura 2.18 ilustra el procesamiento utilizando la *Event Iterator* API. A partir de la instancia de `XMLInputFactory` se crea, en este caso, una instancia de `XMLEventReader`, y ésta se utiliza para recorrer la secuencia de eventos. Al contrario que en el caso anterior, es posible acceder a representaciones explícitas de dichos eventos como objetos. Por lo demás, la estructura de la aplicación es análoga a la anteriormente presentada.

```
import java.util.*;
import javax.xml.stream.*;
import javax.xml.stream.events.XMLEvent;
import java.io.*;

public class StAXXMLEventReaderCurso {
    public List<Asignatura> procesaCurso(String curso) {
        FileReader      reader = null;
        XMLEventReader parser = null;
        String  areaAsignatura  = null;
        String  nombreAsignatura   = null;
        XMLEvent evento   = null;
        String   tag  = null;
        List<Asignatura> asignaturas = new LinkedList<Asignatura>();

        try {
            reader = new FileReader(curso);
            parser  = XMLInputFactory.newInstance().createXMLEventReader(reader);
            while (parser.hasNext()) {
                     evento = parser.nextEvent();
                     if (evento.isStartElement()) {
                          tag=evento.asStartElement().getName().getLocalPart();
                          if ("nombre".equals(tag)){
                                nombreAsignatura = parser.getElementText();
                          } else if ("area".equals(tag)){
                            areaAsignatura = parser.getElementText();
                            asignaturas.append(new Asignatura(nombreAsignatura,
                                                              areaAsignatura);
                          }
                      }

                  }

        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                reader.close();
            } catch (Exception ex){}
            try {
                parser.close();
            } catch (Exception ex){}
        }
     return asignaturas;
    }
}
```

Figura 2.18. Ejemplo de procesamiento StAX usando un estilo *Event Iterator* API.

Para finalizar cabe observar que, al contrario que SAX, StAX facilita el mantenimiento de contexto durante el recorrido del documento. No obstante, las aplicaciones StAX suelen adolecer de menos facilidades de extensibilidad que las aplicaciones SAX. Así mismo, el modelo de información subyacente se formula, también, a nivel de XML en lugar de a nivel de las aplicaciones específicas.

2.3 Herramientas de construcción de procesadores de lenguajes: Generadores de Traductores

En esta Tesis juegan un papel central las *herramientas de construcción de procesadores de lenguaje*, ya que dichas herramientas se proponen como herramientas fundamentales para llevar a cabo el desarrollo de aplicaciones XML. En particular, el tipo de herramientas considerado es el constituido por los *generadores de traductores*. De esta forma, esta sección se va a centrar en el estudio de dichas herramientas de generación de traductores. En el apartado 2.3.1 se revisarán brevemente los fundamentos teóricos de dichas herramientas, mientras que el apartado 2.3.2 presentará ejemplos representativos de las mismas.

2.3.1 Conceptos básicos sobre analizadores sintácticos y traductores dirigidos por la sintaxis

En este apartado se revisan algunos conceptos básicos en los que se fundamentan las herramientas de generación de traductores: gramáticas incontextuales (apartado 2.3.1.1), analizadores ascendentes y descendentes (apartado 2.3.1.2), analizadores LR (apartado 2.3.1.3) y modelos básicos de traducción (apartado 2.3.1.4).

2.3.1.1 Gramáticas incontextuales

Una gramática incontextual es un formalismo que permite describir la estructura sintáctica de un lenguaje informático (Aho et al., 2006), (Grune et al., 2008). Para ello, la gramática se vale de:

- *Símbolos sintácticos* que representan las construcciones sintácticas del lenguaje. Mediante el conjunto de símbolos *terminales* se representan las construcciones simples, y mediante el conjunto de símbolos *no terminales* las construcciones complejas.
- *Reglas sintácticas o producciones* que determinan la estructura de las construcciones sintácticas compuestas (los no terminales) en términos de secuencias de construcciones más simples. Cada no terminal tiene asociado una o varias reglas que definen su estructura, y cada regla está formada por una *cabeza* o *parte izquierda* constituida por el símbolo no terminal, y por el *cuerpo* o la *parte derecha* de la regla. Dicho cuerpo es una secuencia de símbolos terminales y no terminales que define la estructura del no terminal en términos de otras estructuras sintácticas.

Figura 2.19.a) Gramática incontextual, y b) Árbol sintáctico asociado a la sentencia N+N+N.

Las sentencias del lenguaje representado por la gramática incontextual se generan empleando las producciones como reglas de reescritura, de manera que cada símbolo no terminal puede ser sustituido por el cuerpo de alguna de las reglas que tiene asociadas. Este proceso de generación se denomina *derivación*, y siempre comienza por un símbolo no terminal especial que se denomina *axioma* o *símbolo inicial*, y que representa la estructura sintáctica de nivel más alto del lenguaje. El proceso de derivación finaliza cuando ya no hay más no terminales que substituir, de manera que la sentencia que se obtiene es una cadena de terminales. Además este proceso de derivación puede representarse gráficamente en forma de un árbol denominado *árbol de análisis sintáctico*. Este árbol se caracteriza porque los nodos interiores representan símbolos no terminales, los nodos hoja representan símbolos terminales, las relaciones padre – hijo representan la aplicación de una producción (el padre se corresponde con la cabeza de la regla, y los hijos, leídos de izquierda a derecha, se corresponden con el cuerpo de dicha

regla), y por último el nodo raíz representa al axioma de la gramática. En la Figura 2.19 se muestra en a), un ejemplo de gramática, y en b) el árbol de análisis sintáctico asociado a una sentencia del lenguaje representado por la gramática.

El tipo de gramáticas introducidas se denominan frecuentemente *gramáticas* BNF[16] (Backus, 1959). A fin de permitir realizar especificaciones más compactas, es posible generalizar el modelo permitiendo utilizar expresiones regulares arbitrarias sobre los símbolos de la gramática en los cuerpos de las producciones. El formalismo resultante se denomina *gramáticas con partes derechas regulares* (*regular right-part grammars*) o *gramáticas* EBNF[17] (LaLonde, 1976), (ISO, 1996). Dicho formalismo, si bien permite especificaciones más compactas, no aumenta el poder expresivo de las mismas, en el sentido de que toda gramáticas EBNF puede transformarse de manera directa (aunque no única), en una gramática BFN equivalente.

2.3.1.2 Analizadores ascendentes y descendentes

En la clasificación de los analizadores sintácticos, resulta útil pensar en los mismos como constructores de árboles sintácticos en lugar de como meros reconocedores de las cadenas de entrada. De esta forma, una clasificación usual de los analizadores sintácticos es en analizadores *ascendentes* y analizadores *descendentes* (Aho et al., 2006). Dicha clasificación obedece a la forma en la que los analizadores construyen dichos árboles de análisis sintáctico para cada cadena de entrada:

- En los analizadores descendentes, el árbol de análisis sintáctico para una cadena de entrada, se comienza a construir desde el axioma de la gramática, y se expande mientras sea posible una hoja del árbol mediante la aplicación de una producción.
- En cambio, en los analizadores ascendentes, se construye el árbol de análisis sintáctico de una cadena de entrada, a partir de las hojas que representan la cadena de entrada, hasta alcanzar la raíz del árbol que representa al axioma.

Esta concepción es, no obstante, simplemente conceptual, ya que las implementaciones usuales de los analizadores nunca llegan a construir el árbol, por motivos obvios de eficiencia. Por el contrario:

- Durante el análisis de una cadena de entrada w, los analizadores descendentes mantienen una configuración (α, w'), donde w' es la porción de entrada aún no reconocida y α es una cadena de terminales y no terminales mantenida internamente por el reconocedor. Además, considerando que $w = w''w'$, $w''\alpha$ ha de aparecer como paso intermedio en una derivación más a la izquierda de w (es decir, una derivación en la que se rescribe siempre el no terminal más a la izquierda). Estos analizadores realizan dos operaciones básicas: *emparejar* el símbolo actual de la entrada con el primer símbolo de la cadena interna (si éste es un terminal), eliminando dicho símbolo de la entrada y de la cadena interna, y *expandir* el primer símbolo de la cadena (si éste es un no terminal) mediante la aplicación de una producción (Aho et al., 2006), (Grune et al., 2008).

[16] Backus-Naur Form
[17] Extended Backus-Naur Form

- Por su parte, los analizadores ascendentes mantienen, durante el análisis de una sentencia de entrada w, una configuración (α, w'), siendo w', como en el caso descendente, la porción de entrada no reconocida, y α la memoria interna del analizador (mantenida en una *pila de análisis*). Así mismo, en este caso αw' aparece como paso intermedio en una derivación más a la derecha de w (es decir, una derivación en la que siempre se rescribe el no terminal situado más a la derecha). Para ello, los analizadores pueden llevar a cabo dos tipos de operaciones sobre su memoria interna: *desplazar* el terminal actual, anexándolo a la cadena interna, o bien *reducir* una porción final de α, que se corresponde con el cuerpo de una producción de la gramática, por la correspondiente parte izquierda de dicha producción (Aho et al., 2006), (Grune et al., 2008). Por tanto, estos analizadores aplican las producciones en sentido inverso, sustituyendo cuerpos de producciones por sus respectivas cabezas.

El análisis descendente presenta como ventaja la facilidad de implementación, así como su adaptación directa al formalismo EBNF. Como contrapartida, tiene como inconveniente que el número de gramáticas aptas para utilizarse en este tipo de análisis es menor que para los analizadores ascendentes, requiriendo realizar transformaciones sobre las gramáticas para poder usarlas con este método de análisis. Con respecto al análisis ascendente, su principal ventaja es que el número de gramáticas aptas para utilizarse es mucho mayor que en el caso descendente (Aho et al., 2006), y sin necesidad de realizar transformación alguna. Sin embargo presenta como desventaja, la complejidad para añadir *acciones semánticas* en los analizadores (Purdom, 1980) –véase también el apartado 2.3.1.4.

Tal como se ha mencionado, no todas las gramáticas independientes del contexto se pueden utilizar para implementar ambos métodos:

- En el caso de los analizadores descendentes, las gramáticas aptas más extendidas, son las *gramáticas* LL(k). Una gramática LL(k) es aquella que permite seleccionar de forma unívoca las producciones a aplicar en las operaciones de expansión en base a la información proporcionada por los siguientes *k* símbolos de la cadena de entrada que se está analizando (Aho et al., 2006), (Grune et al., 2008).
- De igual forma, en el caso del análisis ascendente, las gramáticas aptas más extendidas son las *gramáticas* LR(k). Una gramática LR(k) es aquella que permite decidir qué operación realizar, reducir o desplazar, en base a la información proporcionada por los siguientes *k* símbolos de la cadena de entrada que se está analizando (Aho et al., 2006), (Grune et al., 2008). En la práctica, el análisis ascendente suele basarse en un tipo de gramática denominado LALR(1) (Aho et al., 2006), (Grune et al., 2008). Esta clase de gramáticas, que se encuentra entre la clase de gramáticas LR(0) y la clase LR(1), permite decidir la operación correcta a realizar utilizando un símbolo de pre-análisis en la mayor parte de los casos presentados en la práctica, amén de facilitar la construcción de analizadores ascendentes más compactos que los obtenidos contemplando todo el potencial de las gramáticas LR(1).

2.3.1.3 Analizadores LR

Una de las arquitecturas más ampliamente utilizada para los analizadores ascendentes es la brindada por los analizadores LR (Aho et al., 2006), (Grune et al., 2008). Dichos analizadores aprovechan la propiedad básica de que las posibles configuraciones de la pila de análisis de un analizador ascendente, *prefijos viables*, conforman un lenguaje regular (Knuth, 1965) para basar el control del analizador en un autómata finito que:

- Reconoce dicho lenguaje de prefijos viables.
- Permite decidir en qué momento aparece el cuerpo de una producción (*asidero*) en la cima de la pila de análisis, de tal forma que sea lícito llevar a cabo una reducción.

Una vez disponible dicho autómata, es posible codificarlo de forma eficiente en una tabla de transición aumentada, en la cuál las transiciones para los terminales (y el fin de fichero) están enriquecidas con acciones (*desplazar*, *reducir*, o *aceptar*). Dependiendo del método utilizado para construir tales autómatas (y las tablas asociadas), será posible conseguir menor o mayor capacidad de análisis, así como mayor o menor eficiencia en la memoria total requerida para almacenar las tablas. De esta forma, existen métodos que, para cualquier gramática LR(*k*), son capaces de construir autómatas reconocedores de prefijos viables, a partir de los cuáles, además, es posible determinar unívocamente cuándo realizar acciones de desplazamiento, y cuándo llevar a cabo acciones de reducción. De estos métodos, los asociados con las gramáticas LR(0) y con las gramáticas LR(1) son especialmente relevantes, ya que para *k*s mayores el número de estados resultante llega a ser prohibitivamente alto.

En este sentido, el método LR(0), que funciona para cualquier gramática LR(0) – es decir, genera autómatas que permiten decidir correctamente los movimientos de análisis siempre y cuando la gramática de entrada sea LR(0), se basa en identificar los estados de los autómatas reconocedores con conjuntos de *elementos* LR(0): elementos de la forma $A \rightarrow \alpha \bullet \beta$, con $A \rightarrow \alpha\beta$ una producción de la gramática. El significado intuitivo de dichos elementos es:

- Es factible estar reconociendo *A*…
- … habiéndose reconocido ya α …
- … y estando pendiente de reconocerse β.

Cada estado quedará completamente definido mediante un conjunto de *elementos nucleares*, obteniéndose el resto de los elementos a partir de los mismos mediante una operación de *cierre*. Dicha operación determina que si $A \rightarrow \alpha \bullet B\beta$ está en un estado *s*, también estarán en *s* todos aquellos elementos de la forma $B \rightarrow \bullet\gamma$. De esta forma:

- Como estado inicial se considera el definido por el cierre de $S' \rightarrow \bullet S$ (S' es un nuevo axioma, S el axioma original, y $S' \rightarrow S$ una nueva regla introducida para facilitar la determinación de la aceptación de las cadenas de entrada).
- Así mismo, supóngase que *s* contiene elementos de la forma $A \rightarrow \alpha \bullet X\beta$, y sean $A \rightarrow \alpha_0 \bullet X\beta_0$ … $A \rightarrow \alpha_0 \bullet X\beta_{n(s)}$ todos los elementos en *s* de dicha forma. Entonces, existirá otro estado *s'* en el autómata con elementos nucleares $A \rightarrow \alpha_0 X \bullet \beta_0$ … $A \rightarrow \alpha_0 X \bullet \beta_{n(s)}$, y una transición de *s* a *s'* etiquetada con X.

El autómata así construido reconoce, efectivamente, los prefijos viables de la gramática original. Así mismo, todo estado en el que exista un elemento de la forma $A\rightarrow\alpha\bullet$ inducirá una reducción por $A\rightarrow\alpha$, independientemente del siguiente símbolo que se observe en la entrada. Este hecho conduce frecuentemente a la aparición de *conflictos* (situaciones en las que más de una operación es posible), que impiden el uso satisfactorio del analizador LR. Tales conflictos pueden ser:

- Conflictos de *desplazamiento – reducción*. En un mismo estado es posible realizar un desplazamiento y una operación de reducción.
- Conflictos de *reducción – reducción*. En un mismo estado es factible reducir por dos reglas distintas.

A fin de evitar dichos conflictos puede enriquecerse los elementos con un *símbolo de* pre-análisis. Como consecuencia se obtiene un método de construcción de tablas de análisis denominado LR-canónico, que funciona para cualquier gramática LR(1). Dicho método considera *elementos* LR(1) de la forma $[A\rightarrow\alpha\bullet\beta,a]$. En estos elementos:

- El elemento $A\rightarrow\alpha\bullet\beta$ se interpreta como en el método LR(0).
- El símbolo *a* se interpreta como que, una vez finalizado el reconocimiento de β, el siguiente símbolo en la entrada ha de ser *a*.

Como en el caso LR(0), los estados del autómata construido por el método quedan determinados por un conjunto de elementos nucleares. La operación de cierre es, ahora, como sigue: si $[A\rightarrow\alpha\bullet B\beta,a]$ está en un estado *s*, $B\rightarrow\gamma$ es una producción, y *b* es un *primero* de $B\beta a$ (i.e., es posible derivar de $B\beta a$ una cadena que comience por *b*), entonces $[B\rightarrow\bullet\gamma, b]$ también estará en *s*. El estado inicial se define a través del cierre de $[S'\rightarrow\bullet S,\$]$ ($ denota el *fin de fichero*), y las transiciones se definen como en el caso LR(0): si *s* tiene un elemento de la forma $[A\rightarrow\alpha\bullet X\beta, a]$ y $[A\rightarrow\alpha_0\bullet X\beta_0, a_0]$ … $[A\rightarrow\alpha_0\bullet X\beta_{n(s)}, a_{n(s)}]$ es el conjunto de todos los elementos de este tipo en *s*, entonces hay un estado *s'* cuyos elementos nucleares son $[A\rightarrow\alpha_0 X\bullet\beta_0, a_0]$ … $[A\rightarrow\alpha_0 X\bullet\beta_{n(s)}, a_{n(s)}]$. Además, habrá una transición etiquetada por X desde *s* a *s'*.

Dado que, en el autómata LR-canónico cada elemento tiene asociado un símbolo de preanálisis, cuando en un estado aparece un elemento $[A\rightarrow\alpha\bullet,a]$ se generará una operación de reducción únicamente en caso de que el siguiente símbolo sea *a*. Con ello, la capacidad de detección de asideros se aumenta espectacularmente (tanto, que el método LR-canónico es virtualmente suficiente para tratar con los lenguajes informáticos más habituales). No obstante, también puede aumentarse sustancialmente el número de estados.

Es por ello que, en la práctica, se utiliza un método intermedio, denominado método LALR. Dicho método consiste en fusionar aquellos estados en el autómata LR-canónico con el mismo *corazón* (el *corazón* de un estado se obtiene eliminando los símbolos de pre-análisis). Como resultado, el autómata resultante tiene el mismo número de estados que el autómata LR(0), pero añade, además, capacidades de predicción. Existen, así mismo, métodos más eficientes de construir este autómata, que evitan la construcción intermedia del autómata LR-canónico (los métodos más usados en la práctica construyen primeramente el autómata LR(0), y, seguidamente, calculan los símbolos de pre-análisis mediante un algoritmo de aproximaciones sucesivas) (Grune et al., 2008). Las gramáticas para las que el método funciona son las ya anteriormente nombradas gramáticas

LALR(1). Si bien existen gramáticas LR(1) que no son LALR(1), tales gramáticas son raras, lo que confiere al método un carácter suficientemente universal.

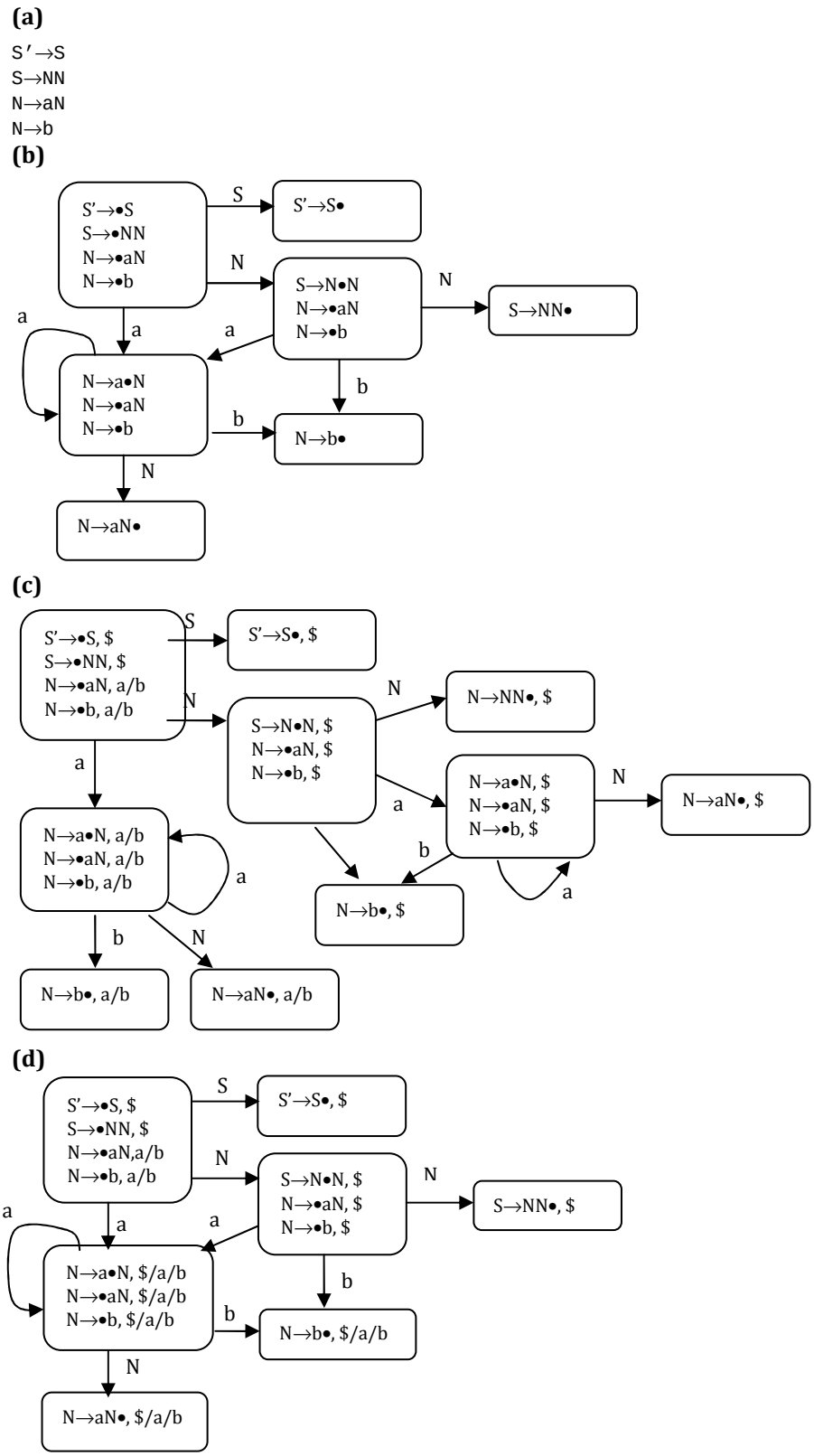


Figura 2.20.a) Una gramática incontextual, b) autómata LR(0) para la gramática en (a), c) autómata LR-canónico, d) Autómata LALR(1).

A fin de ejemplificar los conceptos introducidos anteriormente, la Figura 2.20b, la Figura 2.20c y la Figura 2.20d muestran, respectivamente, los autómatas LR(0), LR – canónico y LALR para la gramática de la Figura 2.20a (en los autómatas LR – canónico y LALR, los elementos $A \rightarrow \alpha$, $a_0/\ldots/a_n$ deben entenderse, realmente, como conjuntos de elementos $[A \rightarrow \alpha, a_0] \ldots [A \rightarrow \alpha, a_n]$).

2.3.1.4 Traductores Ascendentes y Descendentes: Esquemas de Traducción

La incorporación de capacidades de procesamiento a los analizadores supone, añadir *acciones semánticas* que se ejecutan en momentos apropiados del proceso de análisis, y que operan sobre un modelo adecuado de memoria. De esta forma:

- Durante el análisis descendente es posible añadir acciones semánticas que se ejecutan inmediatamente antes / inmediatamente después de cada desplazamiento y expansión[18]. Los artefactos que resultan se denominan *traductores descendentes*. En dichos traductores, las acciones semánticas pueden actuar sobre un estado global, así como sobre un estado local asociado con la aplicación de cada producción. Dependiendo de la manera en la que se implemente el traductor, dicho estado local adoptará una u otra forma. Por ejemplo, en la implementación más habitual (traductores descendentes predictivo-recursivos) (Aho et al., 2006), el estado local queda representado mediante los parámetros y las variables locales de los subprogramas que implementan cada símbolo no terminal. En los traductores dirigidos por tabla, sin embargo, dicho estado se mantiene en *registros semánticos* asociados a cada símbolo y gestionados en una *pila semántica* independiente de la *pila de análisis* (Fischer et al., 1988).
- Durante el análisis ascendente, por otra parte, es posible añadir acciones semánticas con la reducción de cada producción[19], así como utilizar un estado global, o bien registros semánticos asociados con cada símbolo en la pila de análisis. Los artefactos resultantes se denominan *traductores ascendentes*.

De esta forma, aparentemente los traductores ascendentes tienen menor poder de traducción que los ascendentes. Sin embargo, esta apreciación es errónea. Efectivamente, es posible utilizar un pequeño *subterfugio* para *emular* un proceso de traducción descendente durante el análisis ascendente: utilizar no terminales *marcadores* (Watt, 1977). Un no terminal marcador es un nuevo no terminal definido mediante una única producción con cuerpo vacío. De esta forma, dada una producción $A \rightarrow \alpha B \beta$, es posible *simular* la ejecución de una acción antes de comenzar a reconocer B introduciendo un marcador M delante de B (i.e., modificando la producción como $A \rightarrow \alpha M B \beta$). El problema de esta técnica es que la gramática resultante puede perder el carácter poseído por la gramática original - e.g., si la gramática original es LALR(1), la técnica, aplicada indiscriminadamente, no garantiza que la gramática resultante siga siendo LALR(1). No obstante, no es difícil convencerse (Brosgol, 1974) de que el marcado de cada posición de una

[18] En términos de recorridos sobre el árbol de análisis sintáctico subyacente, esto es equivalente a tener la posibilidad de ejecutar acciones semánticas a la entrada y a la salida de cada nodo en un recorrido en preorden de dicho árbol.

[19] En términos de recorridos sobre el árbol de análisis sintáctico subyacente, esto es equivalente a tener la posibilidad de ejecutar acciones semánticas a la salida de cada nodo durante en recorrido en postorden de dicho árbol.

gramática LL(k) con un marcador distinto da como resultado una gramática LL(k), lo que unido al hecho de que toda gramática LL(k) es también una gramática LR(k), demuestra que todo traductor descendente basado en una gramática LL(k) puede ser emulado por un traductor ascendente. Así mismo, la técnica basada en marcadores puede aplicarse también a gramáticas LR(k) que no son LL(k).

Para finalizar, debe indicarse que los traductores pueden especificarse de forma directa enriqueciendo las gramáticas incontextuales con:

- Las acciones semánticas que deben ejecutarse durante el procesamiento. Esto se puede llevar a cabo, por ejemplo, añadiendo fragmentos de código en los sitios apropiados de los cuerpos de las producciones.
- Métodos que permitan referir el estado local en cada producción (e.g., los registros semánticos de cada símbolo del cuerpo, así como el asociado a la cabeza, en los traductores ascendentes).

Los resultados de extender de este modo las gramáticas incontextuales se denominan *esquemas de traducción*, y su naturaleza dependerá, por supuesto, del tipo de traductor a describir.

2.3.2 Herramientas de Generación de Traductores

Los generadores de traductores son las herramientas más ampliamente utilizadas en la implementación de lenguajes informáticos. Este tipo de herramientas toman como entradas esquemas de traducción y generan como salida implementaciones operativas de los traductores especificados. Normalmente estas herramientas generan, bien traductores descendentes, bien traductores ascendentes.

Entre las herramientas generadoras de traductores descendentes, dos de las más populares son JavaCC (*Java Compiler Compiler*) (Kodaganallur, 2004) y ANTLR (*Another Tool for Language Recognition*) (Parr, 2007). Entre las herramientas generadoras de analizadores sintácticos ascendentes, la herramienta clásica es YACC (*Yet Another Compiler Compiler*) (Stephen, 1979). Así mismo, otra herramienta ampliamente utilizada es CUP (*Constructor of Useful Parsers*) (Appel, 1997), (Hudson, 1999). En las siguientes secciones se estudiarán las características principales de estas herramientas[20].

2.3.2.1 Herramientas de Generación de Traductores Descendentes

2.3.2.1.1 JavaCC (Java Compiler Compiler)

JavaCC es un generador de traductores descendentes recursivos escritos en código Java. Para ello, toma como entrada una especificación que amalgama tanto aspectos léxicos (es decir, la definición de los tipos de *tokens* involucrados en el lenguaje), como aspectos sintácticos y de procesamiento. Estos últimos constituyen, de hecho, un esquema de traducción para un traductor descendente predictivo recursivo. De esta forma, las especificaciones JavaCC constan de dos partes bien diferenciadas:

[20] El propósito de estas secciones es ejemplificar las características usuales de este tipo de herramientas, sin pretender realizar una revisión exhaustiva de las distintas herramientas de generación de traductores existentes. Para una compilación exhaustiva puede consultarse, por ejemplo, http://catalog.compilertools.net/.

– Por una parte, se especifican los aspectos léxicos del traductor. Para ello se emplean reglas léxicas, que, mediante expresiones regulares, definen los tipos de *tokens* del lenguaje, patrones auxiliares (precedidos por #), y cadenas *ignorables*, que no deben ser pasadas a la etapa de análisis sintácticos (por ejemplo, *blancos*, *comentarios*, *tabuladores*, etc.).

```
options { LOOKAHEAD=1;}
PARSER_BEGIN(Calculadora)
public class Calculadora {
    public static void main(String args[]) throws ParseException {
      Calculadora parser = new Calculadora(System.in);
      System.out.print("Inserta expresión: ");
      System.out.flush();
   try {
          System.out.println(parser.expr());
       } catch (ParseException x) {
         System.out.println("Expresión errónea!.");
     }
  }
}
PARSER_END(Calculadora)
SKIP :
{ " " | "\n"| "\t" }
TOKEN : { < MAS: "+" > | < MENOS: "-" > | < MUL: "*"> | <DIV: "/">}
TOKEN :{ < CONSTANTE: ( <DIGITO> )+ > |  < #DIGITO: ["0" - "9"] > }
TOKEN : { < PUNTO: "." >}
int expr() :
{int resul, resul2;}
{
    resul = term() ( <MAS> resul2 = term() {resul += resul2;}  |
                     <MENOS> resul2 = term() {resul -= resul2;})*
    {return resul;}
}
int term() :
{int resul, resul2; }
{
    resul=fact() ( <MUL> resul2 = fact() {resul *= resul2;}|
                    <DIV> resul2 = fact() {resul /= resul2;} )*
    {return resul;}
}
int fact() :
{int resul,signo=1;
 Token cte;}
{
    (<MENOS> {signo=-signo;})*
        (cte=<CONSTANTE> {resul =Integer.valueOf(cte.image).intValue();} |
         "(" resul = expr() ")" ) {return signo*resul;}
}
```

Figura 2.21.Ejemplo de traductor en JavaCC.

– Por otra parte, se introducen los aspectos semánticos y de traducción. Dicha especificación recuerda, en gran medida, a un traductor predictivo recursivo codificado manualmente. Efectivamente, cada no terminal se identifica con un método Java. Dicho método introduce una serie de parámetros de entrada, un tipo devuelto, una sección de variables locales, y un cuerpo. Dicho cuerpo define la estructura sintáctica del no terminal, mediante una expresión EBNF, en la que, además, pueden intercalarse las acciones semánticas mediante código Java. La diferencia con una codificación manual estriba en que, ahora, el proceso de decisión entre múltiples opciones es transparente al desarrollador. Efectivamente, JavaCC calculará bajo qué condiciones elegir cada opción posible, mediante un análisis automático de la gramática subyacente.

Obsérvese, por tanto, que, como la mayor parte de las herramientas que generan traductores descendentes recursivos, JavaCC permite definir la estructura de los no terminales mediante producciones EBNF. Así mismo, JavaCC soporta

gramáticas LL(*k*) para cualquier *k* pre-establecido (por defecto, asume únicamente un símbolo de pre-análisis).

La Figura 2.21 muestra un ejemplo de especificación en JavaCC, que implementa un pequeño evaluador de expresiones aritméticas. En dicha especificación:

- Mediante la cláusula `OPTIONS` se especifica el número de *tokens* de pre-análisis a utilizar (opción `LOOKAHEAD`, que, en este caso, se inicializa al número por defecto).
- Entre las cláusulas `PARSER_BEGIN` y `PARSER_END` se encierra el código del programa principal (en general, estas cláusulas podrán encerrar otros métodos y definiciones Java que se añadirán a la clase que implementa el traductor –en este caso, `Calculadora`).
- A continuación se definen las cadenas ignorables (bajo `SKIP`), así como los tipos de *tokens* del lenguaje (bajo `TOKEN`). Obsérvese que las definiciones auxiliares (por ejemplo, `DIGITO`) se preceden por #.
- Por último se especifican los aspectos sintácticos y de procesamiento. La regla `expr` determina la estructura de las expresiones: una cadena de *términos* (no terminal `term`) operados con + o -. La regla para `term` determina que los términos se forman operando factores (`fact`) mediante * o /. Por su parte, `fact` define los factores como una secuencia, posiblemente vacía, de cambios de signo, seguido, bien de una constante, bien de una expresión entre paréntesis.

Obsérvese cómo las correspondientes funciones asociadas con los no terminales devuelven el valor de las correspondientes expresiones, y cómo los valores devueltos por las funciones asociadas a los símbolos del cuerpo se pueden capturar en variables locales. Así mismo, los valores devueltos se pueden computar en acciones semánticas, acciones en las cuáles puede decidirse también los valores devueltos.

```
grammar ExpSimple;
options {
    language=Java;
}
@members {
}
ESPACIO : (' ' | '\n' | '\t')+ { skip(); } ;
MAS: '+';
MENOS: '-';
MUL: '*';
DIV: '/';
CONSTANTE: '0'..'9'+ ;

s        :  a=expr { System.out.println($a.val); };
expr returns [int val] : b=term {$expr.val=$b.val;}
                          ( MAS c=term {$expr.val += $c.val} |
                            MENOS d=term {$expr.val -= $d.val} )* ;
term returns [int val] : a=fact {$term.val=$a.val;}
                          ( MUL b=fact {$term.val *= $b.val} |
                            DIV c=fact {$term.val /= $c.val} )* ;
fact returns [int val] : {$fact.val=1;} ( MENOS {$fact.val=-$fact.val;})*
                             ( CONSTANTE
                                {$fact.val *=
                                   Integer.valueOf(CONSTANTE.text()).intValue();}
                                |
                               '(' expr ')' {$fact.val *= $expr.val;} ) ;
```

Figura 2.22.Ejemplo de traductor en ANTLR.

2.3.2.1.2 ANTLR(Another Tool for Language Recognition)

ANTLR es una herramienta similar a JavaCC, en el sentido de generar también traductores descencentes recursivos, aunque incluye un lenguaje de especificación más expresivo que ofrece, por ejemplo, facilidades para la modularización, así como para ciertos tipos de procesamiento (por ejemplo, construcción de árboles de sintaxis abstracta). La herramienta también soporta características sofisticadas, como soporte para la generación de traductores en múltiples lenguajes. Así mismo, ANTLR soporta un método de análisis denominado LL(*) que permite adaptar el número de símbolos de pre-análisis a utilizar a cada situación, en lugar de utilizar un número de símbolos pre-establecido. Para ello, substituye el uso de conjuntos fijos de símbolos o cadenas para realizar la predicción por autómatas finitos (véase (Parr et al., 2011) para más detalles).

A modo de ejemplo, la Figura 2.22 muestra la especificación ANTLR para el evaluador de expresiones aritméticas sencillas especificado en el apartado anterior.

2.3.2.2 Herramientas de Generación de Traductores Ascendentes

2.3.2.2.1 YACC (Yet Another Compiler Compiler)

YACC es una herramienta de generación de traductores clásica, debido a su asociación con las principales distribuciones de UNIX (Schreiner et al., 1985). YACC permite especificar traductores ascendentes mediante esquemas de traducción, a partir de los cuáles genera traductores ascendentes por desplazamiento-reducción, guiados por tablas, escritos en C (aunque también existen versiones de YACC para virtualmente cualquier lenguaje de programación mínimamente popular).

YACC soporta gramáticas LALR(1), escritas en formato BNF[21]. Cada producción puede tener asociada una acción, que se ejecuta durante la reducción de la misma. En dichas acciones es posible referir a los registros semánticos mediante *pseudovariables*:

- Mediante $$ es posible referir al registro semántico de la cabeza de la producción.
- Mediante $i es posible referir al registro semántico del i-esimo símbolo en el cuerpo de dicha producción.

Así mismo, YACC permite referir a registros semánticos asociados a los símbolos hermano, símbolos *tío*, etc. que preceden al símbolo en la cabeza de la producción, utilizando 0 o desplazamientos negativos en las pseudovariables. Esta característica es muy útil, en combinación con el uso de no terminales marcadores, para acceder a la información de contexto de un símbolo, almacenada en los registros semánticos de su marcador. Por último, YACC permite especificar directamente acciones semánticas en el interior de las producciones (en este caso, crea automáticamente marcadores para dichas acciones), incorporar tratamiento de errores mediante *reglas de error*, y mecanismos extra-gramaticales para resolver problemas de ambigüedad (por ejemplo, especificación de prioridad y asociatividad de operadores).

[21] En realidad, en un EBNF restringido, en el que los cuerpos de las producciones para un mismo no terminal pueden amalgamarse como opciones de una misma regla.

La especificación de la Figura 2.23 muestra la implementación en YACC del evaluador de expresiones considerado en las anteriores secciones. Obsérvese que, al contrario de lo que ocurre con los traductores descendentes, en YACC es posible utilizar gramáticas recursivas a izquierdas para expresar operadores que asocian a izquierdas sin mayores problemas. De hecho, el uso de recursión a izquierdas en YACC (así como en otras herramientas similares) conduce a implementaciones mucho más eficientes en memoria que el uso de recursión a derechas. En particular, para gramáticas en las que la única recursión utilizada es recursión a izquierdas, será posible generar traductores capaces de operar con una pila de análisis acotada.

```
%token CONSTANTE
%%
expr  : expr '+' term {$$ = $1 + $3;} |
         expr '-' term {$$ = $1 - $3;} |
         term {$$=$1;};
term : term '*' fact {$$ = $1 * $3;} |
       term '/' fact {$$ = $1 / $3;} |
       fact {$$=$1;};
fact : '-' fact {$$ = -$2;} |
       atom {$$ = $1; }
atom : CONSTANTE {$$ = $1;} |
       '(' exp ')' {$$ = $2;}
```

Figura 2.23.Ejemplo de especificación en YACC.

Es interesante notar, también, que, al contrario que las herramientas presentadas anteriormente, YACC no incluye mecanismos explícitos para especificar los aspectos léxicos. De esta forma, los traductores generados por YACC se deberán complementar con la implementación de un analizador léxico adecuado (en particular, YACC está especialmente pensado para que los traductores se integren con analizadores léxicos generados automáticamente a partir de especificaciones basadas en definiciones regulares mediante la herramienta LEX (Schreiner et al., 1985)).

Para finalizar, indicar que una herramienta íntimamente relacionada con YACC es BISON (Levine et al. 2009), herramienta desarrollada bajo licencia GNU que, aparte de proporcionar soporte multilenguaje (e.g., permite generar traductores escritos en C, C++ y Java), soporta también otros tipos de gramáticas (e.g., gramáticas incontextuales arbitrarias mediante el método de análisis GLR (Scott et al., 2006)).

2.3.2.2.2 CUP(Constructor of Useful Parsers)

CUP (Appel, 1997) (Hudson, 1999) es un generador de traductores ascendentes para Java y para gramáticas LALR(1) muy parecido a YACC. Así mismo, genera implementaciones razonablemente eficientes en comparación con otras herramientas similares. Por último, incluye un entorno de ejecución orientado a objetos bastante flexible, que permite, por ejemplo, modificar el algoritmo de análisis para llevar a cabo implementaciones más eficientes en aplicaciones especializadas.

Como YACC, CUP soporta también gramáticas en formato BNF. El acceso a los registros semánticos puede realizarse, sin embargo, asociando variables nombradas con los distintos símbolos, lo que, normalmente, mejora la legibilidad de las especificaciones. En relación con el acceso al contexto, que en YACC se llevaba a cabo especificando desplazamientos menores que 1 en las

pseudovariables, CUP no ofrece ningún mecanismo particular. En su lugar, es necesario acceder directamente a la pila semántica del traductor. Como YACC, CUP soporta mecanismos de desambiguado basados en precedencias y asociatividades, así como manejo de errores mediante reglas de tratamiento de error. Así mismo, como en el caso de YACC, los traductores generados mediante CUP deben integrarse con un analizador léxico externo (construido manualmente, o bien generado mediante un generador de analizadores léxicos para Java, como JFLex (Appel, 1997)).

```
/*importaciones*/
import java_cup.runtime.*;
import java.io.*;
/*Declaraciones para el parser*/
   parser code {:
      public static void main(String[] arg){
        /* Crea un objeto parser */
        parser parserObj = new parser();
       /* Asignar el analizador léxico implementado por Yylex */
        Scanner miAnalizadorLexico =
                new Yylex(new InputStreamReader(System.in));
                 parserObj.setScanner(miAnalizadorLexico);
       /*Arrancar el proceso*/
       try{
              parserObj.parse();
             }catch(Exception x){
                   System.out.println("Error fatal.");
             }
          }
        :};
/* Terminales sin valor semántico */
terminal MAS, MENOS, MUL, DIV, PAP, PCIERRE, FIN;
/* Terminales con valor semántico */
terminal Integer CONSTANTE;
/* No terminales sin valor semántico */
non terminal expresion;
/* No terminales con valor semántico */
non terminal Integer expr, term, fact, atom;

/* Gramática */
expresion ::= expr: v {: System.out.println("resultado: "+v);:} FIN;
expr ::= expr: v1 MAS term: v2
{: RESULT = new Integer(v1.intValue() + v2.intValue()); :} |
         expr: v1 MENOS term: v2
{: RESULT = new Integer(v1.intValue() - v2.intValue()); :} |
         term:v {: RESULT=v;:};
term ::= term: v1 MUL fact: v2
{: RESULT= new Integer (v1.intValue() * v2.intValue()); :} |
         term: v1 DIV fact: v2
{: RESULT=new Integer (v1.intValue() / v2.intValue()); :} |
         fact: v {: RESULT=v; :};
fact ::= MENOS fact:v {: RESULT=-v; :} |
         atom:v {: RESULT=v; :};
atom ::= CONSTANTE:v {: RESULT=v; :} |
         PAP expr:v PCIERRE {: System.out.println(v); RESULT=v;:};
```

Figura 2.24.Ejemplo de especificación en CUP.

A modo de ejemplo, la Figura 2.24 muestra la especificación CUP del evaluador sencillo de expresiones utilizado para ilustrar los generadores anteriormente introducidos.

2.4 Gramáticas de atributos

Otro de los pilares básicos de esta Tesis se encuentra en las *gramáticas de atributos*, como técnicas de especificación de procesadores de lenguaje (y, en particular, de traductores) de más alto nivel. Efectivamente, en esta Tesis se propondrá el uso de los modelos básicos de gramáticas de atributos como

mecanismos de especificación de aplicaciones de procesamiento de documentos XML, así como la generación de implementaciones eficientes a partir de dichas especificaciones. Por tanto, en este apartado se revisan los elementos del formalismo de las gramáticas de atributos que resultan más relevantes de cara a esta Tesis. En el apartado 2.4.1 se introduce el modelo básico de las gramáticas de atributos. El apartado 2.4.2 introduce los mecanismos de evaluación semántica asociados al mismo. El apartado 2.4.3 introduce algunas extensiones al formalismo. El apartado 2.4.4 se centra en las extensiones del formalismo sobre gramáticas EBNF, particularmente relevantes en distintas propuestas relacionadas con la realizada en esta Tesis. Para finalizar, el apartado 2.4.5 enumera brevemente algunas herramientas.

```
sent ::= expr where ctes
   expr.defsh = ctes.defs
   sent.val = expr.val
sent ::= expr
   expr.defsh = ∅
   sent.val = expr.val
expr ::= expr OpAd term
   expr(1).defsh = expr(0).defsh
   term.defsh = expr(0).defsh
   expr(0).val = opera(OpAd.op, expr(1).val, term.val)
expr ::= term
   term.defsh = expr.defsh
   expr.val = term.val
term ::= term OpMul fact
   term(1).defsh = term(0).defsh
   fact.defsh = term(0).defsh
   term(0).val = opera(OpMul.op, term(1).val, fact.val)
term ::= fact
   fact.defsh = term.defsh
   term.val = fact.val
fact ::= - fact
   fact(1).defsh = fact(0).defsh
   fact(0).val = - fact(1).val
fact ::= atom
   atom.defsh = fact.defsh
   fact.val = atom.val
atom ::= num
  atom.val = valorDe(num.lex)
atom ::= id
   atom.val = buscaValor(id.lex, atom.defsh)
atom ::= ( expr )
   expr.defsh = atom.defsh
   atom.val = expr.val
OpAd ::= +
   OpAd.op = suma
OpAd ::= -
   OpAd.op = resta
OpMul ::= *
   OpMul.op = multiplica
OpMul ::= /
   OpMul.op = divide
ctes ::= ctes , cte
  ctes(0).defs = ctes(1).defs ∪ {cte.def}
ctes ::= cte
  ctes.defs = {cte.def}
cte  ::= id = num
  cte.def = (id.lex, valorDe(num.lex) )
```

Figura 2.25.Ejemplo de gramática de atributos.

2.4.1 El Modelo Básico

Las gramática de atributos son un formalismo introducido por Donald E. Knuth (Knuth, 1968) para describir la sintaxis y semántica de los lenguajes

incontextuales. Dichas gramáticas extienden las gramáticas incontextuales en el siguiente sentido:

- Cada símbolo de la gramática tiene asociado un conjunto de *atributos semánticos*. Estos atributos se utilizan para dotar de significado a los símbolos, así como para representar la información de contexto necesaria en la determinación de dicho significado.
- Cada producción tiene asociado un conjunto de *ecuaciones semánticas* que indican cómo computar los valores de los atributos en el contexto de dicha producción.

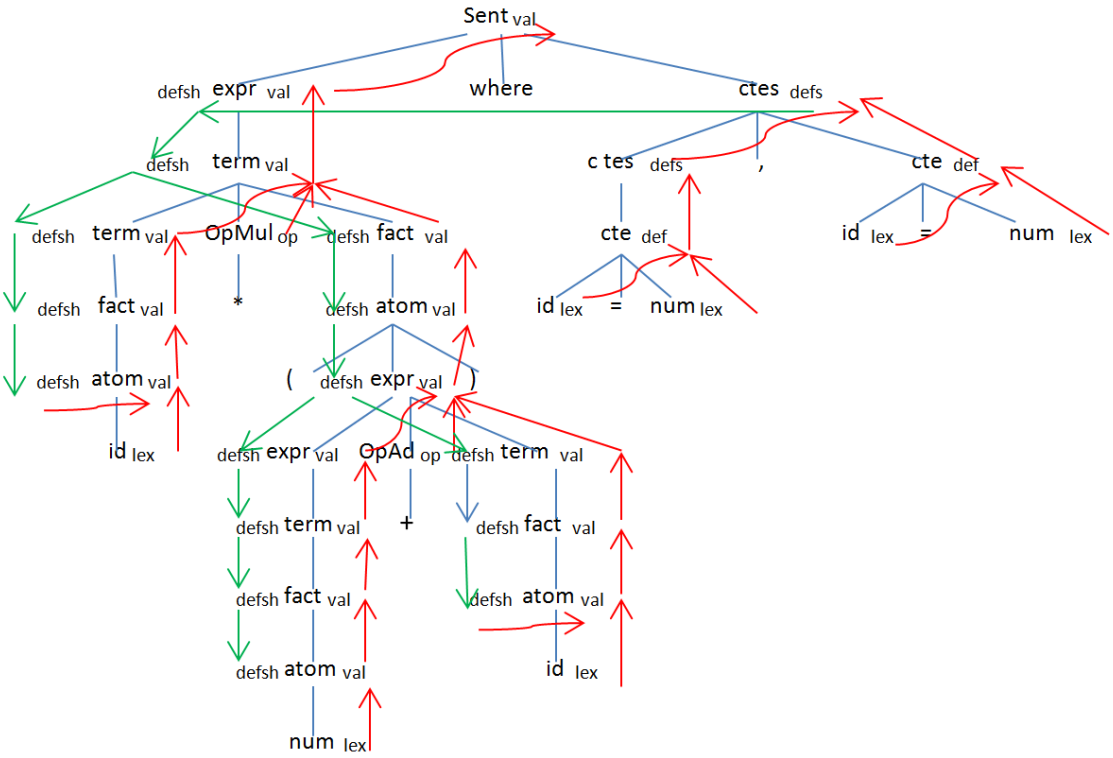


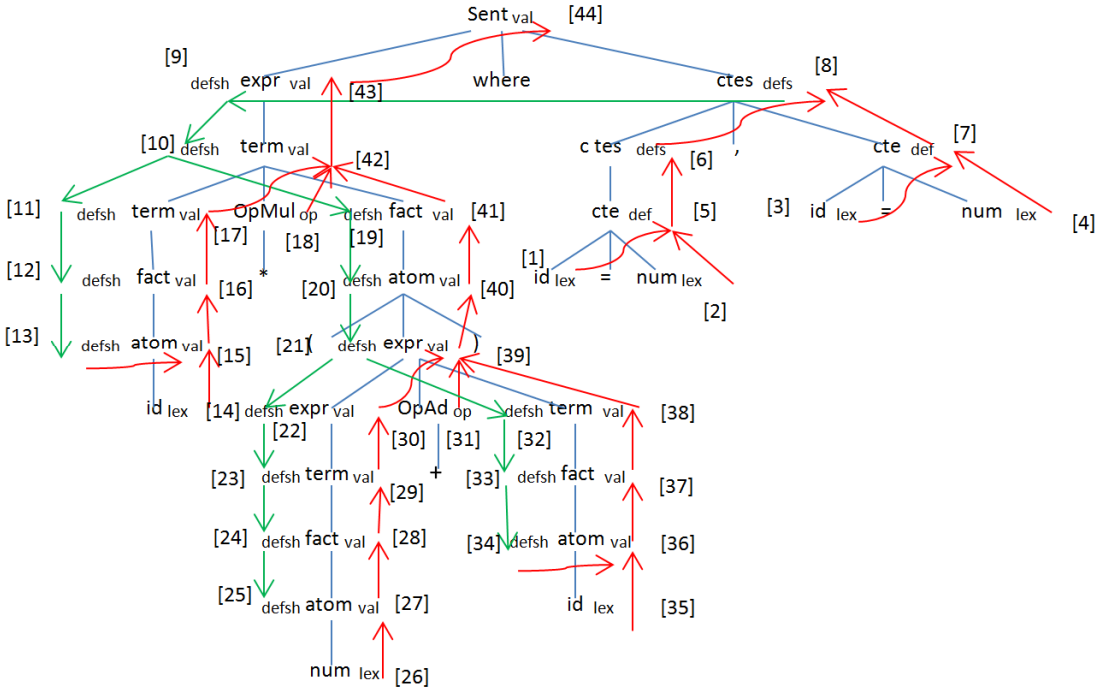
Figura 2.26.Ejemplo de árbol de análisis sintáctico y grafo de dependencias para la sentencia `x*(5+y) where x=-1,y=2.`

De esta forma, desde la perspectiva de los árboles sintácticos, la definición de una gramática de atributos supone asociar instancias de atributos semánticos a los nodos de los árboles, las cuales reciben un valor durante el procesamiento de las sentencias correspondientes de acuerdo a lo indicado por las ecuaciones semánticas.

Los atributos semánticos asociados con un símbolo se dividen, a su vez, en dos tipos, en función de la forma en la que se calcula su valor en el árbol de análisis sintáctico:

- *Atributos sintetizados*. Dichos atributos representan el *significado* asociado al nodo. Su valor se calcula a partir de los valores de los atributos heredados del nodo y de los valores de los atributos sintetizados de los nodos hijos.
- *Atributos heredados*. Dichos atributos representan información de contexto adicional necesaria para determinar el significado del nodo. Su valor se calcula a partir de los valores de los atributos heredados del nodo padre y/o de los valores de los atributos sintetizados de los nodos hermanos.

(a)



(b)

Nodo	Atributo	Valor	Nodo	Atributo	Valor	Nodo	Atributo	Valor
[1]	lex	x	[21]	defsh	{(x,-1),(y,2)}	[41]	val	7
[2]	lex	-1	[22]	defsh	{(x,-1),(y,2)}	[42]	val	-7
[3]	lex	y	[23]	defsh	{(x,-1),(y,2)}	[43]	val	-7
[4]	lex	2	[24]	defsh	{(x,-1),(y,2)}	[44]	val	-7
[5]	def	(x,-1)	[25]	defsh	{(x,-1),(y,2)}			
[6]	defs	{(x,-1)}	[26]	lex	5			
[7]	def	(y,2)	[27]	val	5			
[8]	defs	{(x,-1),(y,2)}	[28]	val	5			
[9]	defsh	{(x,-1),(y,2)}	[29]	val	5			
[10]	defsh	{(x,-1),(y,2)}	[30]	val	5			
[11]	defsh	{(x,-1),(y,2)}	[31]	op	suma			
[12]	defsh	{(x,-1),(y,2)}	[32]	defsh	{(x,-1),(y,2)}			
[13]	defsh	{(x,-1),(y,2)}	[33]	defsh	{(x,-1),(y,2)}			
[14]	lex	x	[34]	defsh	{(x,-1),(y,2)}			
[15]	val	-1	[35]	lex	y			
[16]	val	-1	[36]	val	2			
[17]	val	-1	[37]	val	2			
[18]	op	multiplica	[38]	val	2			
[19]	defsh	{(x,-1),(y,2)}	[39]	val	7			
[20]	defsh	{(x,-1),(y,2)}	[40]	val	7			

Figura 2.27.(a) Orden topológico de los atributos en el grafo de dependencias de la Figura 2.26; (b) Resultado de la evaluación de los atributos.

En este sentido, las ecuaciones semánticas asociadas a cada producción representan cómo llevar a cabo el cálculo del valor de cada atributo sintetizado de

la cabeza y de cada atributo heredado del cuerpo de la regla, mediante la aplicación de *funciones semánticas* a los valores de los atributos heredados de la cabeza y a los atributos sintetizados de los símbolos del cuerpo. De esta forma, el cálculo de cada atributo en cada producción vendrá determinado por una ecuación apropiada, con las dos excepciones siguientes:

- Los atributos de los símbolos terminales, que únicamente pueden ser sinterizados, deberán ser fijados externamente (e.g., por el analizador léxico, durante el análisis léxico de la sentencia de entrada). Dichos atributos se denominan *atributos léxicos*.
- Los atributos heredados del axioma de la gramática deberán también ser fijados externamente (e.g., desde el proceso principal que inicia el procesamiento).

Estas dos excepciones conforman una suerte de *condiciones de contorno* utilizadas en la resolución de las ecuaciones planteadas por la gramática de atributos sobre cada árbol de cada sentencia procesada.

La Figura 2.25 muestra un ejemplo de gramática de atributos, que caracteriza la evaluación de expresiones aritméticas simples, las cuáles pueden ir seguidas de una declaración de constantes. En dicha especificación, el atributo léxico `lex` contiene la cadena concreta del *token* correspondiente. Por su parte, el atributo sintetizado `val` asociado con los distintos tipos de expresiones se utiliza para portar el valor de las expresiones, el atributo sintetizado `defs` asociado con la sección de definiciones se utiliza para construir una tabla que asigna un valor a cada constante, y, por último, el atributo heredado `defsh` se utiliza para propagar dicha tabla a la expresión, a fin de poder determinar los valores de las constantes que en ella se refieren.

En las gramáticas de atributos no es necesario especificar el *orden de evaluación* de las ecuaciones semánticas, pues éste es inducido de las *dependencias* entre atributos definidas por las ecuaciones semánticas: cuando para computar el valor de un atributo se requiere haber computado previamente los valores de un conjunto de atributos entonces se dice que el atributo depende de los atributos del conjunto. En este sentido, todas las dependencias definen un *grafo de dependencias* para cada árbol sintáctico. Dicho grafo (véase la Figura 2.26 para un ejemplo):

- Incluye un nodo por cada atributo de cada nodo del árbol sintáctico.
- Si *b* es atributo de un nodo del árbol sintáctico que depende de un atributo *c*, entonces en el grafo existe un arco desde el nodo asociado a *c* hasta el nodo asociado a *b*.

Así, el orden de evaluación de los atributos debe ser compatible con el grafo de dependencias: es decir los valores de los atributos de los nodos en los árboles sintácticos se computan respetando las dependencias establecidas entre los atributos por las ecuaciones semánticas, de manera que no se puede computar el valor de un atributo si no se han computado antes los valores de todos los atributos de los que éste depende. En este sentido, se trata de un modelo de procesamiento modular, que permite añadir nuevos atributos y ecuaciones sin afectar a los ya existentes, dado que las dependencias se encargarán de reorganizar automáticamente el orden en el que se debe llevar a cabo la evaluación. Esta característica sitúa a las gramáticas de atributos en un nivel de

abstracción más alto que los esquemas de traducción, donde el orden de ejecución de las acciones semánticas se debe especificar explícitamente.

El modelo de evaluación de atributos introducido tiene sentido únicamente para gramáticas *bien formadas*, o *no circulares*, las cuáles no producen grafos de dependencia con ciclos. De esta forma, conceptualmente es posible caracterizar el proceso de evaluación como una *ordenación topológica* de los nodos del grafo de dependencias (véase la Figura 2.27a), y como el cálculo de los valores de los atributos de acuerdo con dicho orden (véase la Figura 2.27b).

2.4.2 Evaluación Semántica

El modelo de evaluación introducido en el apartado anterior es meramente conceptual. En la práctica se emplean mecanismos de evaluación semántica más sofisticados. Dichos mecanismos pueden clasificarse, en un primer nivel, en dos categorías (Ablas, 1991):

- Métodos *estáticos*. Dichos métodos establecen una estrategia de evaluación mediante el análisis de la gramática. Dicha estrategia funcionará para toda posible sentencia. Para ello, dichos métodos imponen restricciones adicionales sobre la gramática, siendo aplicables únicamente a subclases del conjunto de las gramáticas de atributos no circulares. Algunas de estas subclases son las gramáticas *fuertemente no circulares* (Jourdan, 1984), y las gramáticas de atributos *ordenadas* (Kastens, 1980).
- Métodos *dinámicos*. Estos métodos determinan el orden de evaluación más apropiado sobre cada árbol de análisis sintáctico y cada grafo de dependencias particular. De esta forma, tales métodos no restringen el tipo de gramáticas a los que pueden aplicarse, sino que son aplicables a gramáticas no circulares arbitrarias (Cohen et al., 1979).

Los métodos de evaluación dinámicos pueden, a su vez, obedecer a distintas estrategias. Dos de las más frecuentemente usadas son la *evaluación bajo demanda*, y la *evaluación dirigida por los datos*:

- En la *evaluación bajo demanda*, el valor de un atributo se calcula únicamente cuando éste se requiere (Jalili, 1983), (Magnusson et al. 2007). Esto conduce a un modelo de evaluación perezosa, en el que los valores de los atributos se solicitan bajo demanda, conforme se requieren en los distintos cálculos. Como consecuencia, en la evaluación asociada a una determinada sentencia no es necesario computar los valores de todos los atributos, sino únicamente los valores de aquellos atributos requeridos directa o indirectamente por los atributos cuyos valores se ha solicitado. Este modelo permite acomodar de forma natural funciones semánticas *no estrictas*, las cuáles determinan la forma de evaluar sus argumentos, en lugar de seguir una regla de evaluación pre-establecida[22].
- Por su parte, en la *evaluación dirigida por los datos*, el valor de un atributo se calcula tan pronto como los valores de los atributos de los que depende están disponibles (Kennedy et al., 1979). La estrategia de evaluación resultante es, por tanto, impaciente, y computa completamente los valores de todos los

[22] Por ejemplo, la función `if` puede evaluarse como sigue: (i) evalúa su primer argumento, (ii) si éste es cierto, evalúa su segundo argumento, (ii) si es falso, evalúa su tercer argumento

atributos. No obstante, este método puede evitar la construcción completa del árbol de análisis y grafo de dependencias asociado, ya que los atributos pueden liberarse conforme dejan de ser necesitados.

Aparte de estas estrategias de evaluación, un conjunto de estrategias particularmente relevantes en la práctica son las estrategias de evaluación *predeterminadas*. Estas estrategias son un caso particular de las estrategias estáticas, en las que la evaluación de atributos obedece a un conjunto de reglas prefijadas, sin requerir análisis adicional. Para ello, la gramática de atributos debe cumplir ciertas condiciones. Entre éstas, las dos siguientes son especialmente relevantes (véase (Akker et al., 1991)):

- *Gramáticas S-atribuidas*. Son gramáticas de atributos que únicamente incluyen atributos sintetizados. En este tipo de gramáticas es posible computar los valores de los atributos mediante un único recorrido en post-orden del árbol de análisis sintáctico.
- *Gramáticas L-atribuidas*. Son gramáticas de atributos en las que se cumple que en cada producción de la forma $A \rightarrow X_1\ X_2\ ...\ X_n$, cada atributo heredado de X_j con $1 \leq j \leq n$, depende sólo de los atributos sintetizados de los símbolos $X_1\ X_2\ ...\ X_{j-1}$ y de los atributos heredados de A. De esta forma, estas gramáticas permiten una sencilla estrategia de evaluación estática en la que los atributos se evalúan durante un recorrido en pre-orden del árbol de análisis sintáctico, de izquierda a derechas, y preferente en profundidad.

Así mismo, si se imponen restricciones adicionales sobre las gramáticas incontextuales subyacentes, se obtienen clases de gramáticas en las que los valores de los atributos pueden computarse *al vuelo* durante el proceso de análisis sintáctico. Efectivamente, mientras que las anteriores estrategias implican, en mayor o menor medida, la construcción explícita del árbol de análisis sintáctico y/o del grafo de dependencias, en una estrategia de evaluación *al vuelo* los atributos semánticos se computan directamente durante el proceso de análisis sintáctico (Aho et al., 2006). De éstas, las dos siguientes son particularmente relevantes (Wilhelm, 1982):

- *Gramáticas LL-atribuidas*. Son gramáticas L-atribuidas para las cuáles las gramáticas incontextuales subyacentes son gramáticas LL(*k*). Con este tipo de gramáticas es posible evitar la construcción del árbol, almacenar directamente los atributos en parámetros y variables locales de los procedimientos (en un analizador recursivo), o en una *pila semántica* (en uno no recursivo (Fischer et al., 1988)), y computar sus valores durante el análisis (los heredados, justo antes de expandir el correspondiente no terminal, los sintetizados una vez finalizada la expansión de la correspondiente regla).
- *Gramáticas LR-atribuidas*. Este tipo de gramáticas permiten el cálculo de los atributos durante un análisis ascendente. Para ello, las gramáticas subyacentes deben ser de un tipo adecuado de gramáticas LR (e.g., LALR(1), LR(1) o LR(*k*)), y cumplir condiciones adicionales que garanticen que, en cada estado de los correspondientes autómatas, es posible calcular de forma unívoca los atributos heredados que intervienen en las correspondientes posiciones. Algunas de estas gramáticas pueden implementarse utilizando la técnica de marcadores ya aludida anteriormente. Otras requieren generalizar el algoritmo de análisis para permitir asociar la ejecución de acciones con

estados del autómata reconocedor de prefijos viables subyacente (véase (Akker et al., 1990) para una descripción más detalladas de este tipo de gramáticas y de sus subclases).

2.4.3 Extensiones al Modelo Básico

El modelo básico de gramáticas de atributos introducido en la sección 2.4.1 ha sido ampliado, a lo largo de los años, como consecuencia de su aplicación práctica a distintos escenarios de diseño e implementación de lenguajes informáticos (Paakki, 1995). Básicamente, dichas extensiones se pueden clasificar en dos dimensiones distintas:

- Extensiones *organizativas*. Dichas extensiones dotan al lenguaje básico de especificación de características que facilitan su aplicación, tales como facilidades para la modularización y reutilización e inclusión de *patrones de atribución* que definen reglas de propagación de atributos por defecto (Kastens et al., 1994), o inclusión de características de lenguajes de programación generales, tales como orientación a objetos (Hedin, 1989; Hedin, 1999), paradigma lógico (Paakki, 1991), genericidad (Saraiva et al., 1999), orientación a aspectos (Rebernak et al., 2006), etc. Los formalismos resultantes no añaden, no obstante, mayor poder expresivo, en el sentido de que las especificaciones siempre pueden traducirse al modelo básico.
- Extensiones *operacionales*. Dichas extensiones suponen una extensión efectiva de la semántica operacional del modelo básico. Por ejemplo, una de las extensiones más habituales es permitir dependencias circulares entre atributos. Como consecuencia, en las gramáticas *circulares* resultantes el modelo de evaluación semántica cambia, a su vez (en este caso, se utilizan algoritmos de aproximaciones sucesivas –de *punto fijo*- para aproximar iterativamente los valores de los atributos (Jones, 1990)). Otra extensión consiste en dotar al formalismo de *orden superior*, de tal forma que los valores de ciertos atributos pueden interpretarse, a su vez, como árboles de análisis sintácticos atribuidos susceptibles de ser evaluados. El modelo resultante se denomina *gramáticas de atributos de orden superior* (Vogt et al. 1989), y exhibe distintas ramificaciones (véase, por ejemplo (Farrow et al,. 1992)).

2.4.4 Soporte de Notación EBNF

Una extensión del modelo básico de las gramáticas de atributos especialmente relevante en el ámbito del procesamiento de XML es el permitir notación EBNF en la formulación de las gramáticas incontextuales subyacentes. El motivo es que los árboles documentales XML son árboles no acotados (en general, sus tipos de nodos pueden tener un número no acotado de hijos). Por tanto, las gramáticas documentales en XML adoptan, tal y como ya se ha discutido, notación EBNF para describir modelos de contenidos.

Aunque no existe un acuerdo comúnmente aceptado para la extensión del modelo de gramáticas de atributos a EBNF, en las últimas cuatro décadas se han propuesto diversas soluciones, entre las que cabe destacar las siguientes:

- En el sistema basado en gramáticas de atributos ALADIN (Kastens et al., 1982) se dio ya soporte a un tipo restringido de gramáticas EBNF. Las restricciones prohibían iteraciones anidadas en las expresiones regulares, así como

operadores de alternancia, lo que facilitaba la especificación de las reglas de síntesis y herencia de atributos (e.g., síntesis de un atributo como una lista de todos los valores de un determinado atributo en los símbolos del cuerpo de una regla, herencia de un atributo del padre al correspondiente atributo en los hijos, o distribución de los elementos de un atributo tipo lista en el padre al correspondiente atributo en los hijos).

- En (Bochmann, 1976) se proporcionó soporte para expresiones regulares no restringidas en las partes derechas de las producciones. Para ello se establecieron mecanismos que permitían fijar, en cada iteración, la inicialización, redefinición y finalización de atributos, así como la dirección de la evaluación.
- En (Wilhelm, 1982) se propuso un mecanismo restringido a las gramáticas L-atribuidas. Efectivamente, en dichas gramáticas es posible asociar, con cada atributo en una producción, una variable local a dicha producción, e indicar las expresiones a utilizar para calcular los valores de dichas variables en distintos puntos de un recorrido apropiado del árbol de análisis sintáctico (por ejemplo, de izquierda a derechas, y preferente en profundidad). De esta forma, en cada producción EBNF es posible especificar distintas ecuaciones para cada atributo, cada una de las cuáles se registra en un punto adecuado del recorrido (la solución es similar a la utilizada en la herramienta ANTLR).
- Por último, en (Jullig et al., 1984) se propone una extensión del modelo denominado *gramáticas de atributos con parte derecha regular* (*regular right-part attribute grammars*). Al igual que en el caso anterior, dicha extensión permite asociar variables locales con atributos, e impone, así mismo, reglas de ámbito para dichas variables. Así mismo, introduce operadores sofisticados de propagación de valores de atributos (por ejemplo, en la secuencia de nodos generado por una cláusula iterativa: (i) fijar un atributo heredado *h* del primer nodo a un determinado valor, (ii) para los nodos intermedios, fijar *h* al valor de un atributo sintetizado *a* en el nodo precedente, y (iii) tomar el valor de un atributo sintetizado *a*' en el último nodo como el resultado).

Mientras que estas extensiones pueden conducir a especificaciones más compactas, en muchos casos tales especificaciones son también difíciles de entender, sobre todo cuando se emplean expresiones regulares complicadas, que ocultan estructura relevante de cara al procesamiento[23].

2.4.5 Herramientas

El formalismo de las gramáticas de atributos es soportado, así mismo, por diversas herramientas, comenzando por herramientas clásicas como GAG (Kastens et al., 1994) FNC-2 (Jourdan et al., 1991), ELI (Gray et al., 1992), Ox (Bischoff, 1992) o Elegant (Augusteijn, 1990), y finalizando con propuestas más recientes como LISA (Henriques et al., 2005), Silver (Wyk et al., 2010) o JastAdd (Ekman et al., 2007). Muchas de estas herramientas adoptan estrategias de evaluación estática, y, por tanto, restringen el tipo de gramáticas aceptables. Así mismo, la mayoría

[23] Por ejemplo, piénsese en la formulación de procesamientos que tengan en cuenta prioridades y asociatividades de operadores, sobre una sintaxis como E ::= (\+|\-)* (n|id|'('E')') ((\+|\-|*|/)(\+|\-)* (n|id|'('E')'))*.

construyen explícitamente árboles de análisis sintáctico, y utilizan los mismos para dirigir el proceso de evaluación semántica.

2.5 Procesamiento de documentos XML Dirigido por Lenguajes

Las tecnologías de procesamiento de documentos XML estudiadas en el apartado 2.2 se caracterizan porque consideran los documentos XML como estructuras de datos: árboles, secuencias de eventos o ristras de elementos de información. Sin embargo tal como se ha analizado en el apartado 2.1.1, los documentos XML son instancias de un lenguaje XML que viene definido mediante una gramática documental. Así, una aplicación de procesamiento de documentos XML puede ser considerada como un procesador de lenguajes de marcado específicos, y por tanto llevarse a cabo su desarrollo utilizando las herramientas propias para generar procesadores de lenguajes. En esta sección se estudian diversas iniciativas que adoptan en mayor o menor medida esta perspectiva del procesamiento de documentos XML.

2.5.1 Generación de Aplicaciones de Documentos XML a partir de Esquemas de Traducción

Estas propuestas se caracterizan porque utilizan esquemas de traducción para especificar aplicaciones de procesamiento XML. En este contexto, un esquema de traducción consiste en una representación formal de la gramática para un tipo de documento XML, en el que se embeben fragmentos de código de un lenguaje de programación que representan acciones semánticas que deben ejecutarse conforme los elementos del documento XML son reconocidos durante el análisis de los mismos. Los dos sistemas más representativos de esta aproximación son la herramienta ANTXR (Stanchfield, 2005), construida sobre la herramienta de generación de analizadores ANTLR ya introducida anteriormente, y RelaxNGCC (Okajima, 2002), una extensión del lenguaje de esquema RelaxNG. A continuación se describen estos sistemas.

2.5.1.1 RelaxNGCC (Regular Language for XML -Next Generation- Compiler Compiler)

RelaxNGCC (Okajima, 2002) es un generador de traductores para documentos XML construido a partir de RelaxNG. RelaxNGCC toma como entrada una gramática RelaxNG que tiene embebidos fragmentos de código orientado a Java, y genera como salida un traductor codificado en Java, e implementado sobre un marco SAX. Mediante la gramática RelaxNGCC se representa el esquema de un tipo de documento XML, y mediante los fragmentos de código que se embeben se representan las acciones que deben ejecutarse cuando los diferentes elementos del tipo representado por la gramática son reconocidos. Así, el traductor generado en Java implementa el reconocimiento del documento XML y las acciones a ejecutar durante su reconocimiento. Las gramáticas aceptadas por RelaxNGCC deben

permitir un reconocimiento basado en autómatas sobre árboles deterministas (Comon et al., 1997) [24].

Con el objetivo de ejemplificar el uso de RelaxNGCC, se va a considerar nuevamente la DTD de la Figura 2.11, que representaba un sub-lenguaje de marcado para el ejemplo de las asignaturas de un curso. La Figura 2.28 muestra la especificación RelaxNGCC que implementa el procesamiento que permite crear una lista con los nombres y áreas de cada asignatura. En dicha especificación, el elemento `java-import` permite importar clases utilizadas en las acciones semánticas. Por su parte, `java-body` permite añadir campos adicionales a la clase Java generada por RelaxNGCC (dicha clase implementa la interfaz `ContentHandler` de SAX). El atributo `alias` de la etiqueta `text` permite identificar, mediante nombres, las porciones del documento necesarias (nombre de asignatura, y área). Por último, las acciones se incluyen como contenidos del elemento `java.`

```
<?xml versión="1.0" encoding= "utf-8">
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:cc="http://www.xml.gr.jp/xmlns/relaxngcc">
<start  cc:class="ListaAsignaturas">
  <cc:java-import>
   import java.util.*;
  </cc:java-import>
  <cc:java-body>
    ArrayList lista = new ArrayList();
  </cc:java-body>
   <element name="curso">
      <oneOrMore>
          <element name="asignatura">
            <element name="nombre"><text cc:alias="nombre"/></element>
            <element name="area"> <text cc:alias="area"/></element>
            <cc:java>
              lista.add(nombre);
              lista.add(area);
            </cc:java>
          </element>
      </oneOrMore>
   </element>
</start>
</grammar >
```

Figura 2.28.Gramática *RelaxNG* con etiquetas *RelaxNGCC* embebidas.

De esta forma, RelaxNGCC permite implementar aplicaciones de procesamiento de documentos XML genéricas de acuerdo con un enfoque dirigido por lenguajes. No obstante, su principal desventaja es acoplar dicho procesamiento con la gramática documental, en lugar de utilizar gramáticas independientes, y, posiblemente, más adaptadas a cada necesidad concreta de procesamiento.

2.5.1.2 ANTXR (Another Tool for XML Recognition)

ANTXR (Stanchfield, 2005) es un generador de traductores para documentos XML, que toma como entrada una gramática ANTXR, y que genera como salida un traductor implementado en Java. Una gramática ANTXR es una gramática ANTLR adaptada para representar el esquema de un tipo de documento XML, y que permite embeber fragmentos de código java que representan las acciones semánticas que deben realizarse cuando se reconocen los elementos de un documento XML del tipo representado por la gramática. El traductor que genera

[24] RelaxNGCC reconoce internamente una gramática RELAXNG utilizando un autómata determinista conducido por eventos SAX

ANTXR lleva a cabo el reconocimiento del documento XML y la ejecución de las acciones semánticas definidas. De esta forma, ANTXR puede concebirse como un preprocesador para ANTLR orientado a describir esquemas de traducción que permiten procesar documentos XML.

Para ilustrar el uso de ANTXR, se va a considerar nuevamente el ejemplo de las asignaturas de un curso, y la creación de una lista que recopile las asignaturas y sus nombres. La Figura 2.29 muestra la correspondiente especificación ANTXR. Obsérvese que dicha especificación es análoga a una especificación ANTLR, con la salvedad de que se incluye azúcar sintáctico orientado al dominio del procesamiento de documentos XML (en el ejemplo, inclusión de etiquetas, de tal manera que es posible identificar etiquetas con no terminales, y asociar reglas en consonancia).

```
header {
package com.javadude.antlr.sample.xml;

import java.util.List;
import java.util.ArrayList;
}

class CursoParser extends Parser;

document returns [List results=null]
  : results=<curso> EOF;

<curso> returns [List results=new ArrayList()]
{Asignatura a;}
  : (a=<asignatura> {results.add(a);})+
  ;

<asignatura> returns [Asignatura a=new Asignatura()]
 {String nombre,area;}
  : nombre= <nombre> { a.setNombre(nombre);}
    area= <area> { a.setArea(area);}
  ;

<nombre> returns [String value=null]
  : pcdata:PCDATA {value=pcdata.getText();}
  ;

<area> returns [String value=null]
  : pcdata:PCDATA {value=pcdata.getText();}
  ;
```

Figura 2.29. Gramática *ANTXR* con acciones semánticas.

ANTXR supone, de esta forma, otro ejemplo que muestra la factibilidad de entender las aplicaciones de procesamiento de documentos XML como tipos particulares de procesadores de lenguaje. En concreto, el hecho de estar construido sobre una herramienta ampliamente utilizada para la construcción de traductores refuerza dicha percepción. No obstante, ANTXR asume algunos convenios (en concreto, la identificación de etiquetas con no terminales) que suponen cierto acoplamiento entre la estructura del documento y la estructura demandada por el procesamiento, que pueden limitar su aplicabilidad a escenarios en los que dichas estructuras difieren.

2.5.2 Procesamiento de Documentos XML a partir de gramáticas de atributos

Tal como se estudio en el apartado 2.4, las gramáticas de atributos constituyen un mecanismo formal para especificar el procesamiento dirigido por una gramática incontextual. En este sentido, las especificaciones formales de los lenguajes XML

poseen una naturaleza gramatical que podría ser utilizada como base para la descripción de procesamientos mediante gramáticas de atributos. Sin embargo, hay que tener en cuenta que:

- Las gramáticas documentales (DTDs, Esquemas XML, etc.) soportan notaciones EBNF, mientras que las gramáticas de atributos se formulan, en principio, sobre gramáticas BNF. Por tanto, los modelos básicos de las gramáticas de atributos no son, como ya se ha discutido, directamente aplicables para describir el procesamiento de un documento XML.
- Existen similitudes entre las gramáticas de atributos y las especificaciones formales de un lenguaje XML. Por ejemplo los elementos XML podrían equipararse con símbolos no terminales de una gramática incontextual, la definición de dichos elementos podría equiparse a las reglas que describen la estructura de dicho no terminal, y la asignación de atributos a cada elemento XML podría equiparse con un tipo restringido de ecuaciones semánticas. Sin embargo la noción de función semántica de las gramáticas de atributos no tiene equivalencia en el contexto de las especificaciones formales XML, siendo imposible gestionar atributos heredados y sintetizados, ni tampoco especificar procesamiento alguno sobre los documentos XML.

Las propuestas descritas en esta sección plantean aprovechar la naturaleza gramatical que presenta una especificación formal de un lenguaje XML, y sus similitudes (y diferencias) con las gramáticas de atributos, con el objetivo de describir procesamientos sobre los documentos XML. En la sección 2.5.2.1 se describe un enfoque que plantea no modificar la DTD o esquema original del lenguaje XML y asociar externamente los atributos y ecuaciones semánticas que definen su procesamiento. En la sección 2.5.2.2 se muestra un enfoque que plantea llevar a cabo una transformación de la gramática EBNF del lenguaje documental a una gramática BNF, de forma que sea posible formular una gramática de atributos directamente sobre la gramática transformada. En la sección 2.5.2.3 se describen los enfoques que extienden las gramáticas de atributos al ámbito de las gramáticas EBNF como substrato estructural, de manera que puedan utilizarse directamente sobre las gramáticas EBNF subyacentes a las DTDs o esquemas originales de los lenguajes XML. Por último en la sección 2.5.2.4 se estudian los enfoques que se centran en llevar a cabo transformaciones sobre los árboles documentales asociados a los documentos XML, para obtener nuevos árboles con características más adecuadas para llevar a cabo su procesamiento utilizando gramáticas de atributos.

2.5.2.1 Desacoplamiento de la gramática y las ecuaciones semánticas

Un primer enfoque para describir el procesamiento de documentos XML mediante gramáticas de atributos consiste en describir externamente a las gramáticas documentales (por ejemplo, en otros documentos XML) la semántica del procesamiento en forma de reglas semánticas que definen atributos XML para los elementos de los documentos del lenguaje. En estos enfoques, las gramáticas documentales originales permanecen, por tanto, inalteradas. Su principal limitación, sin embargo, es que la asociación de reglas semánticas con nodos en el árbol documental es muy simple, obviando aspectos estructurales implícitos en los modelos de contenido.

Dos trabajos representativos de este enfoque son los descritos en (Psaila et al., 1999) y en (Havasi, 2002). Tales trabajos se revisan a continuación.

2.5.2.1.1 SRD (*Semantic Rule Definition*)

En (Psaila et al., 1999) se introduce SRD *(Semantic Rule Definition*), un metalenguaje XML para describir el procesamiento de documentos XML mediante gramáticas de atributos. La Figura 2.30 muestra la DTD de este lenguaje.

SRD permite asociar con los tipos de elementos de un lenguaje XML nuevos atributos, denominados *atributos intensionales*, para diferenciarlos de los *atributos extensionales* definidos por la gramática documental. Los valores de dichos atributos no están restringidos a los valores XML básicos, sino que pueden tomar valores de tipos más complejos, tales como registros y conjuntos. Así mismo, tales valores vendrán determinados por la aplicación de *reglas semánticas* apropiadas. De esta forma, SRD ofrece mecanismos para:

```
<!ELEMENT field EMPTY >
<!ATTLIST field name CDATA $REQUIRED
               type CDATA #REQUIRED >
<!ELEMENT recordtype (field)+ >
<!ATTLIST recordtype name CDATA #REQUIRED>
<!ELEMENT settype EMPTY >
<!ATTLIST settype name CDATA #REQUIRED
                 type CDATA #REQUIRED >
<!ELEMENT attribute EMPTY >
<!ATTLIST attribute name CDATA #REQUIRED
                   type CDATA #REQUIRED >
<!ELEMENT add-to-element (attribute)+ >
<!ATTLIST add-to-element name CDATA #REQUIRED >
<!ELEMENT derive EMPTY >
<!ATTLIST derive attribute CDATA #REQUIRED
                 from CDATA #REQUIRED >
<!ELEMENT static ((derive)+) >
<!ELEMENT ex EMPTY >
<!ATTLIST ex element CDATA #REQUIRED >
<!ELEMENT isfirst EMPTY >
<!ATTLIST isfirst element CDATA #REQUIRED >
<!ELEMENT islast EMPTY >
<!ATTLIST islast element CDATA #REQUIRED >
<!ELEMENT not (ex|isfirst|islast) >
<!ELEMENT if ((ex|isfirst|islast|not)+,(derive)+) >
<!ELEMENT conditional ((if)+,else,(derive+) >
<!ELEMENT rules-for ((static)?,(conditional)*) >
<!ATTLIST rules-for element CDATA #REQUIRED
                    father-of CDATA #REQUIRED >
<!ELEMENT types (recordtype | settype)+ >
<!ELEMENT intensional-attributes (add-to-element)* >
<!ELEMENT semantic-rules (rules-for)+ >
<!ELEMENT SRD ((types)*,intensional-attributes,semantic-rules) >
```

Figura 2.30. DTD del metalenguaje SRD (tomada de (Psaila et al., 1999)).

- Definir nuevos tipos para los atributos intensionales (elementos *recordtype* y *settype* en la Figura 2.30).
- Asociar atributos intensionales con elementos (elemento *intensional-attributes* en la Figura 2.30)
- Asociar paquetes de reglas semánticas a pares de nodos *padre-hijo* en el documento. Para ello utiliza bloques *de asociación* (elemento *rules-for* en la Figura 2.30). Cada regla, por su parte, puede ser, bien *estática* (coindice con las ecuaciones semánticas en el modelo básico de las gramáticas de atributos), o *condicional* (su aplicación depende de una condición sobre la estructura de los hijos del elemento padre, lo que permite utilizar unas u otras reglas semánticas dependiendo del contenido particular de tales elementos). En la

DTD de la Figura 2.30, el elemento `static` permite incorporar las reglas estáticas, mientras que el elemento `conditional` permite incorporar las reglas condicionales. Cada regla básica se describe mediante el elemento *derive*.

– Indicar las expresiones a utilizar para computar los valores de los atributos. Para ello, SRD permite referir valores de otros atributos (tanto extensionales como intensionales), construir valores de tipos complejos (conjuntos, registros), e invocar a funciones externas. Tales expresiones se describen utilizando un pequeño lenguaje de expresiones embebido como valor del atributo `from` en los elementos `derive`.

```
<?xml versión=”1.0”?> <!DOCTYPE SRD SYSTEM “srd.dtd”>
<types>
  <recordtype name=”nodo”>
    <field name=”nombre”  type=”string”/>
    <field name=”area”  type=”string”/>
  </recordtype>
  <settype  name=”lista” type=”nodo”>
</types>
<intensional-attributes>
 <add-to-element name=”curso”>
   <attribute name=”lista” type=”lista”/>
 </add-to-element>
<add-to-element name=”asignatura”>
   <attribute name=”nodo” type=”nodo”/>
   <attribute name=”nombre” type=”nombre”/>
   <attribute name=”area” type=”area”/>
 </add-to-element>
<add-to-element name=”nombre”>
   <attribute name=”lista” type=”nombre”/>
 </add-to-element>
<add-to-element name=”area”>
   <attribute name=”lista” type=”area”/>
 </add-to-element>
</intensional-attributes>
<semantic-rules>
 <rules-for element=”DOCUMENT” father-of=”curso”>
  <static> <derive attribute=”curso.lista” from=”#EMPTYSET(nodo)”/></static>
 </rules-for>
<rules-for element=”curso” father-of=”asignatura”>
  <static> <derive attribute=”asignatura.nodo”
                   from=”#RECORD(asignatura.nombre, asignatura.area)”/> </static>
  <static> <derive attribute=”curso.lista” from=”#ADDTOSET(asignatura.nodo)”/></static>
 </rules-for>
<rules-for element=”asignatura” father-of=”nombre”>
  <static> <derive attribute=”asignatura.nombre” from=”nombre.nombre)”/></static>
 </rules-for>
<rules-for element=”asignatura” father-of=”area”>
  <static> <derive attribute=”asignatura.area” from=”area.area”/></static>
 </rules-for>
<rules-for element=”nombre” father-of=”#PCDATA”>
  <static> <derive attribute=”nombre.nombre” from=”#PCDATA.Content”/></static>
 </rules-for>
<rules-for element=”area” father-of=”#PCDATA”>
  <static> <derive attribute=”area.area” from=”#PCDATA.Content”/></static>
 </rules-for>
</semantic-rules>
```

Figura 2.31.Descripción SRD para el ejemplo de la Figura 2.11.

La Figura 2.31 muestra una especificación SRD para la tarea de construcción de la lista de asignaturas y áreas ya descrita anteriormente.

2.5.2.1.2 SRML (*Semantic Rule Metalanguage*)

En (Havasi, 2002) se describe una propuesta similar a la SRD: el metalenguaje *SRML (Semantic Rule Meta Language)*.

La Figura 2.32 muestra la DTD de SRML. Como dicha DTD sugiere, la propuesta es más simple que la de SRD. En particular, SRML no contempla la definición de tipos complejos. Por su parte, los atributos se asocian directamente a partir de la asociación de las reglas semánticas. Dichas reglas, por su parte, se asocian con elementos, en lugar de con pares de nodos *padre – hijo,* como en SRD. Por su parte, las expresiones semánticas en SRML se describen explícitamente mediante elementos, en lugar de mediante un lenguaje específico embebido, como en SRD. Aparte de estas diferencias, ambas propuestas coinciden en lo básico: describir el procesamiento mediante un documento XML independiente, en el que se asocian atributos y reglas semánticas con los elementos de los documentos a procesar.

```
<!ELEMENT semantic-rules (rules-for*)>
<!ELEMENT rules-for(rule*)>
<!ATTLIST rules-for root NMTOKEN #REQUIRED>
<!ELEMENT rule (expr)>
<!ATTLIST rule element NMTOKEN #REQUIRED
              attrib NMTOKEN#REQUIRED >
<!ELEMENT expr(binary-op | attribute | data |
               no-data | if-element | if-expr | if-all |
               if-any | current-attribute | position |
               external-funtion)>
<!ELEMENT binary-op (expr,expr)>
<!ATTLIST binary-op op (add | sub | mul | div | exp | equal | not-equal |
                        less | greater | or | xor | and | nor | contains |
                        concat | begins-with | ends-with) #REQUIRED >
<!ELEMENT attribute EMPTY>
<!ATTLIST attribute element NMTOKEN "srml:this"I
                     num NMTOKEN "O"
                     from (begin | current | end) current
                     attribe NMTOKEN #REQUIRED >
<!ELEMENT if-element (expr,expr)>
<!ATTLIST if-element from (begin | end) "begin">
<!ELEMENT position EMPTY>
<!ATTLIST position element NMTOKEN "srml:all"
                   from (begin | end) "begin" >
<!ELEMENT if-all (expr,expr,expr)
<!ATTLIST if-all element NMTOKEN "srml:all"
                 attrib NMTOKEN "srml:all" >
<!ELEMENT if-any (expr,expr,expr)>
 <!ATTLIST if-any element NMTOKEN "srml:all"
                  Attrib NMTOKEN "srml all" >
<!ELEMENT current-attribute EMPTY>
<!ELEMENT if-expr (expr,expr,expr)>
<!ELEMENT data (#PCDATA)>


<!ELEMENT no-data EMPTY>
<!ELEMENT extern-funtion (param)*>
<!ATTLIST extern-funtionname NMTOKEN #REQUIRED>
<!ELEMENT param (expr)>
```

Figura 2.32. DTD del metalenguaje SRML (tomada de (Havasi, 2002)).

A modo de ejemplo, la Figura 2.33 muestra la especificación SRML de la tarea relativa a asignaturas y actividades.

2.5.2.2 Enfoques basados en transformación de gramáticas

Una de las principales dificultades para aplicar el modelo de gramáticas de atributos al procesamiento de documentos XML es que, mientras que las gramáticas incontextuales subyacentes a las gramáticas de atributos son gramáticas BNF, las gramáticas documentales son gramáticas EBNF. La sección anterior abordó el problema estableciendo mecanismos simples para asociar ecuaciones con nodos en los árboles documentales. No obstante, como se comentó anteriormente, este enfoque es demasiado restrictivo en lenguajes con modelos de

contenido complejos. En esta sección se introduce otro enfoque, basado en transformar la gramática documental en una gramática BFN equivalente.

Los trabajos descritos en (Gançarski et al., 2002), (Gançarski et al., 2006) constituyen un ejemplo típico de este enfoque de transformación de gramáticas. Efectivamente, como se describe en (Gançarski et al., 2002), estos trabajos utilizan un conjunto predefinido de reglas de transformación que permiten obtener una gramática BNF a partir de una DTD. Cada regla se corresponde con un tipo de declaración de elemento o con la aplicación de un operador en la expresión regular de los contenidos de los elementos que aparecen especificados en la DTD. Más concretamente, algunas de las reglas indicadas en (Gançarski et al., 2002) son:

– Para cada elemento *ElemType* cuyo modelo de contenidos es una secuencia de N elementos (es decir, <!ELEMENT ElemType (Comp_1,…,Comp_N)> se crea una producción de la forma *ElemType→ Comp_1…Comp_N*

```
<semantic-rules>
<rules-for  root=”curso“>
<rule element=”srml:root” attrib=”lista“>
<expr>
 <extern-function name=”añadir-lista”>
   <param> <expr>
       <attribute element=”srml:root “ num=1 from=”current“ attrib=”lista“>
     </expr></param>
   <param <expr>
       <attribute element=”asignatura“ num=1 from=”current“ attrib=”nodo“>
     </expr></param>
</expr>
</rules-for>
<rules-for  root=”asignatura“>
<rule element=”srml:root” attrib=”nodo“>
<expr>
 <extern-function name=”crear-nodo”>
   <param> <expr>
       <attribute element=”nombre“ num=1 from=”current“ attrib=”nombre“>
     </expr></param>
   <param> <expr>
       <attribute element=”area“ num=1 from=”current“ attrib=”area“>
     </expr></param>
</expr>
</rules-for>
<rules-for  root=”nombre“>
<rule element=”srml:root” attrib=”nombre“>
  <expr>
    <attribute element=”srml:root“ num=1 from=”current“ attrib=”nombre.pcdata“> </expr>
</rules-for>
<rules-for  root=”area“>
<rule element=”srml:root” attrib=”area“>
  <expr>
    <attribute element=”srml:root“ num=1 from=”current“ attrib=”area.pcdata“/> </expr>
</rules-for>
</semantic-rules>
```

Figura 2.33. Descripción SRML para el ejemplo de la Figura 2.11.

– Para cada elemento declarado como una alternativa de N elementos (es decir, *<!ELEMENT ElemType (Comp_1|…|Comp_N)>*) se crea una producción por cada elemento optativo, *ElemType→ Comp_1 … ElemType→ Comp_N*

– Para elementos *ElemType* cuyo modelo de contenidos viene dado por una expresión regular más compleja, se crea una producción *ElemType → NewSymbol*, donde *NewSymbol* es la raíz de la gramática que representa la expresión regular.

Así mismo, algunas de las reglas indicadas en (Gançarski et al., 2002) para transformar modelos de contenidos en gramáticas son:

- Para *(E +)*, se crean las producciones $NewSymbol_{(E+)} \rightarrow NewSymbol_{E}$ $NewSymbol_{(E+)}$ y $NewSymbol_{(E+)} \rightarrow NewSymbol_{E}$, donde $NewSymbol_{E}$ es el axioma de la gramática para *E*, y $NewSymbol_{(E+)}$ es un nuevo no terminal introducido para *(E +)*
- Análogamente, para *(E *)* se crean las producciones $NewSymbol_{(E^*)} \rightarrow NewSymbol_{E}$ $NewSymbol_{(E^*)}$ y $NewSymbol_{(E^*)} \rightarrow \lambda$, y para *(E ?)* las producciones $NewSymbol_{(E?)} \rightarrow NewSymbol_{E}$ y $NewSymbol_{(E?)} \rightarrow \lambda$.

(a)

```
Curso → AuxAsignatura
AuxAsignatura →  Asignatura AuxAsignatura
AuxAsignatura →  Asignatura
Asignatura → Nombre Area
Nombre → TEXT
Area → TEX
```

(b)

```
Curso → AuxAsignatura
 Curso.lista= AuxAsignatura.lista
AuxAsignatura →  Asignatura AuxAsignatura
 AuxAsignatura0.lista= añadir-lista(Asignatura1.lista, Asignatura.nodo)
AuxAsignatura →  Asignatura
 AuxAsignatura.lista = añadir-lista(crea-lista(), Asignatura.nodo)
Asignatura → Nombre Area
 Asignatura.nodo= crea-nodo(Nombre.contenido, Area.contenido)
Nombre → TEXT
 Nombre.contenido= TEXT.text
Area → TEXT
 Area.contenido= TEXT.text
```

Figura 2.34. (a) Resultado de transformar la DTD en la Figura 2.11, (b) gramática de atributos construida sobre (a).

La aplicación de reglas como las descritas permite obtener una gramática BNF que representa la estructura sintáctica del lenguaje XML descrito en la DTD. Sobre dicha gramática será posible, entonces, definir gramáticas de atributos que especifiquen distintos procesamientos de los documentos.

Con el objetivo de ilustrar esta propuesta, la Figura 2.34(a) muestra la aplicación de las reglas de transformación a la DTD de la Figura 2.11. Sobre dicha gramática puede, entonces, añadirse atributos y ecuaciones para llevar a cabo la síntesis de la lista de asignaturas y áreas, tal y como se muestra en la Figura 2.34(b).

Para finalizar, es interesante observar que, si bien esta propuesta resuelve la discrepancia EBNF – BNF que surge al aplicar el modelo de gramáticas de atributos al procesamiento de documentos XML, su principal problema es la definición de un conjunto predefinido de reglas de transformación sobre gramáticas, que se aplican independientemente del tipo de tarea a resolver. De esta forma, las estructuras impuestas por las gramáticas resultantes sobre los documentos serán apropiadas en algunas ocasiones, pero inapropiadas en otras ocasiones.

2.5.2.3 Enfoques basados en gramáticas de atributos sobre EBNF

A fin de abordar la discrepancia BNF – EBNF existente entre el modelo básico de las gramáticas de atributos y las gramáticas documentales, es posible acomodar al procesamiento de documentos XML algunas de las propuestas de extensión a EBNF del modelo básico descritas en el apartado 2.4. Dos propuestas representativas de estos esfuerzos son la de las *Gramáticas de Atributos Extendidas* (Neven, 1999),

(Neven, 2005) y la de las *Gramáticas de Atributos Orientadas a Flujos* XML (Koch et al., 2007). Dichas propuestas se analizan a continuación.

2.5.2.3.1 Gramáticas de Atributos Extendidas

Las *Gramáticas de Atributos Extendidas* (EAGs – *Extended Attribute Grammars*) propuestas en (Neven, 1999) (Neven, 2005) fueron diseñadas como mecanismo de ejecución eficiente de consultas sobre documentos XML. Dichas gramáticas pueden considerarse como un refinamiento del modelo de las gramáticas de atributos con parte derecha regular a las que se ha hecho alusión en el apartado 2.4.4.

En una EAG, las ecuaciones involucran los siguientes componentes:

- Un *selector de valores de atributos*. La evaluación de dicho selector sobre un contexto (un nodo en el árbol documental) da lugar a una lista de valores construida a partir de los valores de los atributos en dicho contexto.
- Un *predicado de aplicabilidad*. Dicho predicado actúa sobre el resultado del selector de atributos, y determina si la ecuación es o no aplicable en el contexto.
- Una *función semántica*. Dicha función se aplica sobre el resultado del selector para proporcionar el valor del atributo al que está asociada la ecuación.

En el modelo propuesto en (Neven, 1999) (Neven, 2005), tales componentes se concretan como sigue:

- El selector de atributos se especifica como una secuencia de referencias a atributos en la producción. De esta forma, la lista de valores resultante se obtiene considerando en pre-orden los nodos del contexto, y concatenando, para cada nodo, los valores de los atributos indicados en el selector (en el orden indicado). Para asegurar que cada nodo en el contexto se corresponde unívocamente con un símbolo en la producción, las partes derechas de las producciones deben ser expresiones regulares *no ambiguas* (Brüggemann-Klein et al., 1998)[25].
- El predicado de aplicabilidad es una expresión regular sobre cierto conjunto finito *D*. De esta forma, los atributos están limitados a tomar valores en *D*.
- Por último, la función semántica es, simplemente, una constante.

```
Curso → Asignatura+
   Curso.descripciones =
      <[1].descripcion>, λv.true, λv.v>
Asignatura → Nombre Area
   Asignatura.descripcion =
     <[1].nombre,[2].area>, λv.true, λv.(v[0],v[1])>
Nombre → ([a-z]|[A-Z]|' ')+
   Nombre.nombre =
     <[1].texto,[2].texto,[3].texto>, λv.true, λv.v>
Area → ([a-z]|[A-Z]|' ')+
   Area.nombre =
     <[1].texto,[2].texto,[3].texto>, λv.true, λv.v>
```

Figura 2.35. Ejemplo de EAG.

De esta forma, cada atributo relevante en cada producción puede tener asociada una o varias ecuaciones. En tiempo de evaluación, el predicado de aplicabilidad se utiliza para determinar la ecuación correcta a aplicar en cada contexto. Es

[25] Informalmente, una expresión regular *r* es no ambigua cuando, dado cualquier símbolo *X* de cualquier cadena *w* perteneciente al lenguaje denotado por *r*, siempre es posible identificar una única posición en *r* capaz de producir *X* en *w*.

interesante observar también que, aunque las restricciones impuestas por (Neven, 1999) (Neven, 2005) son suficientes para el dominio restringido de consulta perseguido, el modelo de las EAGs es generalizable para permitir expresar otros muchos tipos de procesamiento.

A modo de ejemplo, la Figura 2.35 ilustra la construcción de una lista con las asignaturas y las áreas utilizando una EAG. En dicha EAG, el selector se especifica según la propuesta de (Neven, 1999) (Neven, 2005). Para ello, la posición del símbolo en el que se selecciona el atributo se especifica entre corchetes (por ejemplo, *[1].descripcion* en *Curso* → *Asignatura*+ selecciona el atributo *descripcion* de todos los elementos *Asignatura* en *Curso*). El predicado, por su parte, se especifica como una función sobre los valores seleccionados (en este ejemplo, todos los predicados se evalúan a *true*, ya que existe una única ecuación para cada atributo)[26]. Por último, la función semántica se especifica también como una función sobre dicho valor (en el cuerpo de la función, el valor se trata como un *array*).

2.5.2.3.2 Gramática de Atributos Orientadas a Flujos XML

Las *Gramáticas de Atributos Orientadas a Flujos XML* (XSAGs – XML *Stream Attribute Grammars*) son una propuesta descrita en (Koch et al., 2007) para el procesamiento eficiente de flujos XML que puede considerarse como un refinamiento de las gramáticas L-atribuidas con gramáticas EBNF subyacentes a las que se ha hecho alusión en el apartado 2.4.4 (Wilhelm, 1982). Más concretamente, esta propuesta considera dos modelos de complejidad creciente:

- El modelo bXSAG (*Basic* XSAG), en el que únicamente los elementos pueden contener atributos.
- El modelo yXSAG (*easY* XSAG), en el que tanto los elementos como cada subexpresión en sus modelos de contenidos pueden tener asociados atributos.

De esta forma, el proceso de evaluación de atributos en las gramáticas bXSAG se lleva a cabo sobre los árboles documentales, mientras que en las gramáticas yXSAG dichos árboles se enriquecen, además, con la estructura asociada a los árboles de sintaxis abstracta de las expresiones regulares que representan los modelos de contenido.

El modelo de evaluación en este tipo de gramáticas puede explicarse adecuadamente considerando que cada nodo *n* tiene un único atributo heredado *h*, y un único atributo sintetizado *s*[27]. De esta forma:

```
Curso → {λv.(⊥,⊥,⊥,[])} (Asignatura +)
Asignatura → Nombre Area {λ(_,N,A,D).(⊥,⊥,⊥,D++[(N,A)])}
Nombre → #PCDATA {λ(T,_,_,D).(⊥,T,⊥,D)}
Area → #PCDATA {λ(T,N,_,D).(⊥,N,T,D)}
```

Figura 2.36. Ejemplo de gramática yXSAG.

- El valor de *h* en *n* se computa a partir del valor del atributo *h* del nodo padre, en caso de que *n* sea el primer hijo, o bien a partir del atributo *s* del nodo

[26] La notación λv.(…) es la notación lambda habitual para funciones anónimas.

[27] Dado que puede considerarse que los valores de dichos atributos son *arrays*, la simplificación no resta generalidad a la discutida en (Koch et al., 2007), pero sí que permite una explicación del proceso de evaluación mucho más intuitiva y sencilla.

hermano que lo precede, en otro caso.

– El valor de *s* en *n*, por su parte, se computa a partir del valor de *s* del último hijo, en caso de que *n* tenga nodos hijo, o bien a partir del valor del nodo *h* de *n*, en otro caso.

(a)

```
<curso>
   <Asignatura>
     <Nombre>Autómatas</Nombre>
     <Area>Computación</Area>
   </Asignatura>
   <Asignatura>
     <Nombre>Lógica</Nombre>
     <Area>Matemáticas</Area>
   </Asignatura>
 </curso>
```

(b)

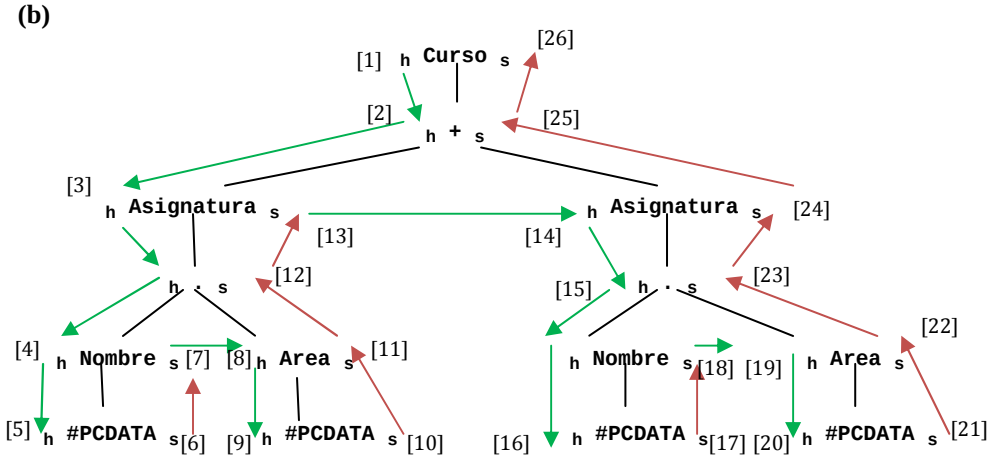


Figura 2.37. Ilustración del modelo de evaluación asociado a las gramáticas yXSAG (parte 1): (a) documento de ejemplo, (b) árbol de análisis sintáctico y grafo de dependencias inducido por la gramática en la Figura 2.36.

Para computar dichos valores se aplica, por defecto, una función identidad. Este comportamiento puede modificarse indicando una *función de atribución* adecuada:

– Inmediatamente antes de la subexpresión que genera el nodo, para el caso del atributo *h*.

– Inmediatamente después de la subexpresión (en caso de nodos que se correspondan con expresiones regulares), o al final de la producción (para el caso de nodos correspondientes a elementos), para el caso del atributo *s*.

Con estas restricciones, es posible llevar a cabo una evaluación *al vuelo* de los atributos semánticos, pudiendo liberar el espacio necesario para sus instancias tan pronto como los valores de las mismas dejen de ser necesarios.

A fin de ilustrar esta propuesta, la Figura 2.36 muestra una gramática yXSAG para el ejemplo de las asignaturas y áreas de conocimiento. En dicha gramática, los valores de los atributos son 4-tuplas de la forma (*Texto*, *Nombre*, *Area, Descripción*). Se asume, así mismo, que, una vez reconocido un contenido textual #PCDATA, se actualiza el valor (*T, N, A, D*) del atributo *h* a (*T', N, A, D*), con *T'* el citado contenido textual. Así, por ejemplo, la función de atribución $\lambda(\_,N,A,D).(\bot,\bot,\bot,D{+}{+}[(N,A)])$ se aplica tras el reconocimiento de una asignatura para concatenar el par *(N,A)* con el nombre y el área a la lista de asignaturas recolectada hasta el momento. Así mismo, en los puntos en los que no se indica

función de atribución alguna, se asume que se aplica la función de atribución por defecto $\lambda v.v$. La Figura 2.37 y la Figura 2.38 ilustran el proceso de evalución inducido por esta gramática sobre un documento de ejemplo. Dicha figura hace palpable cómo las gramáticas yXSAG pueden concebirse como formas de transformar un flujo de información por defecto que fluye a través de los atributos *h* y *s* en un recorrido de izquierda a derecha, y preferente en profundidad, del árbol documental enriquecido con las sintaxis abstractas de los modelos de contenido.

Nodo	Atributo	Valor	Nodo	Atributo	Valor
[1]	h	(⊥,⊥,⊥,⊥)	[14]	h	(⊥,⊥,⊥,[(Autómatas,Computación)])
[2]	h	(⊥,⊥,⊥,⊥)	[15]	h	(⊥,⊥,⊥,[(Autómatas,Computación)])
[3]	h	(⊥,⊥,⊥,[])	[16]	h	(⊥,⊥,⊥,[(Autómatas,Computación)])
[4]	h	(⊥,⊥,⊥,[])	[17]	s	(⊥,Lógica,⊥,[(Autómatas,Computación)])
[5]	h	(⊥,⊥,⊥,[])	[18]	s	(⊥,Lógica,⊥,[(Autómatas,Computación)])
[6]	s	(⊥,Autómatas,⊥,[])	[19]	h	(⊥,Lógica,⊥,[(Autómatas,Computación)])
[7]	s	(⊥,Autómatas,⊥,[])	[20]	h	(⊥,Lógica,⊥,[(Autómatas,Computación)])
[8]	h	(⊥,Autómatas,⊥,[])	[21]	s	(⊥,Lógica,Matemáticas,[(Autómatas,Computación)])
[9]	h	(⊥,Autómatas,⊥,[])	[22]	s	(⊥,⊥,⊥,[(Autómatas,Computación), (Lógica,Matemáticas)])
[10]	s	(⊥,Autómatas, Computación,[])	[23]	s	(⊥,⊥,⊥,[(Autómatas,Computación), (Lógica,Matemáticas)])
[11]	s	(⊥,⊥,⊥, [(Autómatas,Computación)])	[24]	s	(⊥,⊥,⊥,[(Autómatas,Computación), (Lógica,Matemáticas)])
[12]	s	(⊥,⊥,⊥, [(Autómatas,Computación)])	[25]	s	(⊥,⊥,⊥,[(Autómatas,Computación), (Lógica,Matemáticas)])
[13]	s	(⊥,⊥,⊥, [(Autómatas,Computación)])	[26]	s	(⊥,⊥,⊥,[(Autómatas,Computación), (Lógica,Matemáticas)])

Figura 2.38. Ilustración del modelo de evaluación asociado a las gramáticas yXSAG (parte 2): resultado de evaluar los atributos en el grafo de la Figura 2.37.

2.5.2.4 Enfoques basados en transformaciones de árboles

En (Nishimura et al., 2005) se describe un enfoque basado en gramáticas de atributos para la transformación eficiente de flujos XML. Básicamente, dicho enfoque se basa en:

- Especificar una transformación, mediante una gramática de atributos, que transforme el flujo XML en una representación de árboles XML canónica apropiada.
- Especificar la transformación sobre dicha representación canónica de árboles XML con una segunda gramática de atributos.
- Especificar una última transformación de la representación canónica de árboles al flujo XML resultante.

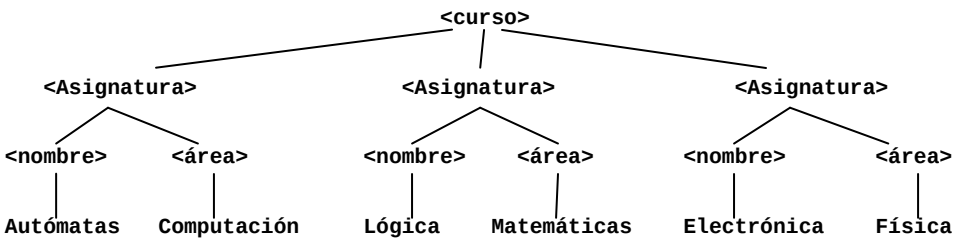
De esta forma, la primera y última gramática son genéricas y reutilizables para cualquier transformación, mientras que la segunda gramática será específica de cada transformación particular. Así mismo, en (Nishimura et al., 2005) se propone utilizar un método de composición de gramáticas de atributos descrito en

(Ganzinger et al., 1984) para derivar automáticamente transformadores de flujos a flujos, que evitan la construcción explícita de árboles, supuestas ciertas restricciones sobre la gramática que especifica la transformación (la segunda de las gramáticas aludidas). Por su parte, en (Nakano, 2004) se relajan las citadas restricciones mediante el uso de un modelo computacional teórico para gramáticas de atributos denominado *transductores de árbol atribuidos*[28] *(ATT)* (Fülop, 1981), (Fülop et al., 1998).

(a)

```
<curso>
   <Asignatura>
     <Nombre> Autómatas </Nombre>
     <Area> Computación </Area>
   </Asignatura>
   <Asignatura>
     <Nombre> Lógica </Nombre>
     <Area> Matemáticas </Area>
   </Asignatura>
   <Asignatura>
     <Nombre> Electrónica </Nombre>
     <Area> Física </Area>
   </Asignatura>
 </curso>
```

(b)



(c)

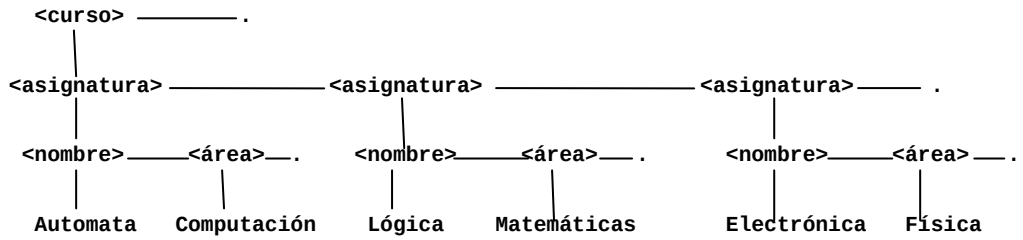


Figura 2.39. Ilustración de la representación de árboles documentales propuesta en (Nishimura et al., 2005) (Nakano, 2004): (a) documento de ejemplo, (b) árbol documental para el documento en a), (b) representación como árbol binario.

Desde el punto de vista del procesamiento genérico de documentos XML, los enfoques propuestos en (Nishimura et al., 2005) (Nakano, 2004) pueden verse como la adopción de una representación canónica para los árboles XML, sobre la cuál puedan formularse gramáticas de atributos al uso, junto con transformaciones entre flujos XML y dicha representación. Dicha representación debe seleccionarse de tal forma que se garantice el funcionamiento del método de composición utilizado. En concreto, (Nishimura et al., 2005) (Nakano, 2004) proponen utilizar

[28] Es un modelo teórico de ejecución orientado a árboles para especificaciones dirigidas por la sintaxis.

una representación *binaria*, acotada, de los árboles XML. Dicha representación involucra dos tipos de nodos distintos:

- Nodos de *contenido*, que almacenan el contenido textual del documento.
- Nodos de *elemento*, que representan elementos en el documento. Un nodo de *elemento* tiene asociada la etiqueta de dicho elemento, así como un máximo de dos hijos. El primero de los hijos apunta, bien al contenido del elemento, bien al primer elemento hijo del mismo, y el segundo apunta al siguiente hermano del elemento (en caso de que tal hermano exista).

(a)

```
Curso → Asignatura
Asignatura → $asignatura Nombre Asignatura
Asignatura → $asignatura Nombre
Nombre → $nombre #PCDATA Area
Area → $area #PCDATA
```

(b)

```
Curso → Asignatura
  Curso.asignaturas = Asignatura.asignaturas
Asignatura → $asignatura Nombre Asignatura
  Asignatura(0).asignaturas = Nombre.asignatura ++ Asignatura(1).asignaturas
Asignatura → $asignatura Nombre
  Asignatura.asignaturas = [ Nombre.asignatura ]
Nombre → $nombre #PCDATA Area
  Nombre.asignatura = (#PCDATA.texto, Area.nombre)
Area → $area #PCDATA
   Area.nombre = #PCDATA.texto
```

Figura 2.40. (a) Gramática para la representación binaria de los documentos sobre asignaturas; (b) especificación de la tarea de construcción de listas de asignaturas sobre la gramática en (a).

La Figura 2.39 muestra un ejemplo de este tipo de representación. Este tipo de representación puede modelarse mediante gramáticas BNF, sobre las que pueden formularse gramáticas de atributos[29]. La Figura 2.40(a) muestra un ejemplo de gramática BNF que caracteriza los documentos de las asignaturas y áreas en esta representación. Por su parte, la Figura 2.40(b) muestra la especificación de la tarea de recolectar la lista de asignaturas y áreas mediante una gramática de atributos.

Para finalizar, indicar que la ventaja de estos enfoques es la posibilidad de derivar automáticamente procesadores eficientes a partir de especificaciones orientadas a árboles. No obstante, la desventaja es la rigidez impuesta por la representación binaria predefinida de los árboles XML, lo que conduce a gramáticas y especificaciones *no naturales*, tal y como deja constancia la especificación en la Figura 2.40(b).

[29] En (Nishimura et al., 2005) (Nakano, 2004) se propone una gramática BNF genérica. No obstante, es posible, y más conveniente desde el punto de vista del procesamiento, generar gramáticas específicas para cada tipo de documento, que, posteriormente, puedan transformarse en términos de la gramática genérica propuesta en (Nishimura et al., 2005) (Nakano, 2004).

2.6 A Modo de Conclusión

Las tecnologías de procesamiento de documentos XML más extendidas promueven un enfoque orientado a datos. De esta forma, los documentos se transforman en estructuras de datos apropiadas, sobre las cuáles se formulan los posteriores procesamientos. Este hecho no aprovecha, no obstante, la naturaleza lingüística del marcado, naturaleza que se deriva del hecho de que XML promueve la definición de vocabularios de marcas específicos mediante gramáticas formales. De esta forma, tales tecnologías promueven:

- Por una parte, soluciones orientadas a resolver tareas de procesamiento muy concretas, de manera que su especificidad limita la posibilidad de aplicación a otras tareas de procesamiento diferentes para las que fueron diseñadas.
- Por otra, soluciones de procesamiento de propósito general, presentadas habitualmente como marcos de procesamiento embebidos en un lenguaje de propósito general que permiten resolver cualquier tarea de procesamiento. Estas últimas presentan como ventaja con respecto a las primeras que el dominio de tareas de procesamiento a las que son aplicables es más amplio. Sin embargo su principal desventaja es que la implementación de la tarea de procesamiento se realiza a un bajo nivel, dado que se manipulan los documentos XML desde los lenguajes de programación en forma de estructuras de datos, dificultando de esta forma el mantenimiento y extensibilidad de las aplicaciones de procesamiento.

Como contrapartida a estas tecnologías, se han descrito propuestas que aprovechan la naturaleza lingüística de los lenguajes de marcado para permitir construir aplicaciones de procesamiento a un nivel más alto de abstracción, entendiendo dichas aplicaciones como procesadores para los lenguajes de marcado utilizados para estructurar los documentos. De esta forma:

- Se han descrito propuestas basadas en esquemas de traducción para la especificación de aplicaciones de procesamiento XML. Tales propuestas introducen metalenguajes específicos que permiten describir esquemas de traducción específicamente orientados al procesamiento de documentos XML.
- Se han descrito, por otra parte, propuestas al procesamiento de documentos XML basadas en gramáticas de atributos. Tales propuestas abren la puerta a especificaciones de más alto nivel que las basadas en esquemas de traducción. Así mismo, aunque algunas de estas propuestas se formularon inicialmente con un propósito específico en mente (normalmente como mecanismo eficiente de ejecución de consultas en documentos), se ha mostrado también cómo tales propuestas pueden generalizarse de manera directa para soportar procesamientos más amplios y generales.

De esta forma, en esta Tesis se toma tales propuestas lingüísticas como punto de partida para mostrar cómo es posible abordar la construcción de aplicaciones de procesamiento de documentos XML como un tipo particular de procesadores de lenguaje. Al contrario que los enfoques analizados en este capítulo, que normalmente presentan un fuerte acoplamiento con las gramáticas documentales, en esta Tesis se propugnará el uso de gramáticas incontextuales independientes de las gramáticas documentales, especialmente adaptadas a las distintas tareas y requisitos de procesamiento. Como consecuencia:

- Se propondrán mecanismos que permitan utilizar herramientas de generación de traductores convencionales (e.g., JavaCC, CUP, …) en la construcción de aplicaciones de procesamiento de documentos XML. Con ello se persigue mostrar, por una parte, que no es necesario recurrir a herramientas específicas para XML a fin de orquestar de forma razonable un proceso de desarrollo de aplicaciones de procesamiento XML dirigido por lenguajes. Y, por otra parte, se persigue mostrar como los enfoques resultantes son notoriamente más flexibles que los promovidos por las citadas soluciones especializadas.
- Se propondrá el uso de gramáticas de atributos formuladas sobre gramáticas incontextuales específicas de cada tarea de procesamiento como un método de especificación de alto nivel de tareas de procesamiento de documentos XML, y se mostrará como, a partir de dichas especificaciones, es posible generar automáticamente implementaciones razonablemente eficientes de las aplicaciones especificadas.

Capítulo 3: Antecedentes, Objetivos y Planteamiento del Trabajo

Como se ha descrito en los capítulos anteriores, el problema del procesamiento de documentos XML cobra relevancia desde el momento en el que la propia especificación no establece norma alguna acerca de cómo llevar a cabo dicho procesamiento, dejando abierto este aspecto a diferentes soluciones. En este sentido, tal y como se ha hecho patente en el capítulo anterior, se han propuesto diversos enfoques a este problema, algunos centrados en abordar procesamientos específicos y por tanto no aplicables a otros escenarios, y otros centrados en abordar cualquier tipo de procesamiento y basados en extender funcionalmente algún lenguaje de propósito general. Así mismo, en la mayoría de los enfoques más convencionales es complicado tanto la generación como el mantenimiento de las aplicaciones, dado que estos procesos se realizan a un bajo nivel de abstracción (normalmente a nivel de código). La característica común que comparten muchos de estos enfoques es el concebir los documentos XML como simples contenedores de datos, por lo que ofrecen un conjunto de servicios que permite manipular los datos que se encuentran embebidos en dichos documentos. A fin de elevar el nivel de abstracción en el desarrollo de las aplicaciones de procesamiento XML, tratando, con ello, de facilitar dicho proceso de desarrollo, es posible aprovechar la naturaleza lingüística en la que se fundamentan los lenguajes de marcado que sirven de base a los documentos XML, adoptando, por tanto, enfoques al desarrollo que beben directamente del campo de la implementación de lenguajes informáticos. Esta Tesis se enmarca dentro de dichos enfoques dirigidos por lenguajes.

De esta forma, en este capítulo se comienza presentando los antecedentes en los que se apoya la Tesis planteada, para, a continuación, establecer los objetivos de la misma. Seguidamente se plantea un enfoque lingüístico para el desarrollo de aplicaciones de procesamiento de documentos XML que se vale de herramientas de construcción de procesadores de lenguaje convencionales (Sarasa et al., 2008a) (Sarasa et al., 2009e) (Sarasa et al., 2012a). Así mismo, dicho enfoque se refina mediante un método de especificación de tareas de procesamiento XML basado en

gramáticas de atributos (Sarasa et al., 2012b). El método se implementa mediante un entorno de desarrollo denominado XLOP (Sarasa et al., 2009a). Por último, el enfoque lingüístico propuesto se aplica a distintos casos de estudio reales (Sarasa et al., 2009b) (Sarasa et al., 2009c) (Sarasa et al., 2009d) (Sarasa et al., 2011) (Sarasa et al., 2012a) (Sarasa et al., 2012b).

3.1 Antecedentes de la Tesis

El presente trabajo de Tesis puede concebirse como la continuación de los esfuerzos previos sobre uso de técnicas lingüísticas para el procesamiento de documentos XML desarrollados en la última década en la UCM, y que actualmente constituyen una de las principales líneas de investigación del grupo de investigación ILSA. Dichos trabajos parten de la Tesis doctoral del Prof. José Luis Sierra "Hacia un Paradigma Documental de Desarrollo de Aplicaciones" (Sierra, 2004), en la que se propuso un método de desarrollo de aplicaciones denominado ADDS (Aproximación Documental al Desarrollo de Software), basado en:

- La definición de lenguajes de marcado para estructurar los documentos que describen las características más relevantes de una aplicación intensiva en contenidos.
- La descripción de las aplicaciones mediante documentos XML estructurados con dichos lenguajes.
- La generación de las aplicaciones mediante el procesamiento automático de dichos documentos.

Con el fin de construir los generadores de las aplicaciones, ADDS introdujo un método de construcción modular de dichos generadores denominado OADDS (Operacionalización en ADDS). Dicho método consistía, básicamente, en un marco para la construcción modular de traductores dirigidos por los árboles documentales. Tales traductores utilizaban, así mismo, los nodos de los árboles para almacenar los resultados de la traducción en atributos alojados en los mismos. En (Sierra et al., 2004), (Sierra et al., 2005) y (Sierra et al., 2006b) se encuentra desarrollado tanto ADDS como su método de operacionalización asociado, OADDS.

Así mismo, en (Sierra et al., 2008a) se describe una evolución de OADDS hacia un enfoque basado en gramáticas de atributos para procesar documentos XML, precursor de la propuesta desarrollada en esta Tesis. En dicho trabajo las gramáticas de atributos se especifican directamente como clases Java, en las cuáles los atributos se representan mediante métodos, y se asocian con tipos de elementos mediante anotaciones.

3.2 Objetivos de la Tesis

Continuando la línea de trabajo sobre procesamiento de documentos XML desarrollada ya en la UCM, el objetivo general de esta Tesis es proponer un enfoque lingüístico para la implementación de aplicaciones que procesan documentos XML basado en el uso de herramientas de desarrollo de procesadores de lenguajes. Este enfoque se fundamenta en el análisis crítico de otros enfoques

diferentes al mismo problema realizado en el capitulo anterior, donde se han identificado las principales características y limitaciones de estas soluciones, y supone el siguiente eslabón en la citada línea de investigación en la UCM. De esta forma, este objetivo general se plasma en dos objetivos concretos:

- El primer objetivo se centra en demostrar que es factible y práctico construir aplicaciones de procesamiento de documentos XML combinando herramientas de generación de analizadores sintácticos / traductores convencionales con marcos de trabajo estándares para el procesamiento XML.
- El segundo objetivo se centra en demostrar que es factible y ventajoso especificar declarativamente tareas de procesamiento de documentos XML utilizando gramáticas de atributos, y que partiendo de estas especificaciones de alto nivel es posible generar las aplicaciones que implementan el procesamiento descrito.

Las dos siguientes subsecciones profundizan en cada uno de estos aspectos, sentando las bases para el planteamiento del trabajo de Tesis, que se esboza en la siguiente sección.

3.2.1 Formular un enfoque lingüístico para el desarrollo de aplicaciones de procesamiento de documentos XML basado en herramientas convencionales de construcción de procesadores de lenguaje

Una de las principales características de XML es su carácter *descriptivo*. Efectivamente, los lenguajes de marcado inducidos por XML establecen una separación explícita entre la estructura de la información contenida en el documento XML y la forma en la que se procesa dicha información. Este hecho se debe a que las marcas definidas en los lenguajes de marcado no permitan especificar cómo se debe procesar la información contenida en el documento XML. Por el contrario, y tal y como ya se ha indicado anteriormente, este aspecto está fuera del alcance del metalenguaje. Esto ha llevado a definir todo un repertorio de propuestas que abordan el aspecto del procesamiento, las más relevantes de las cuáles han sido revisadas en el capítulo anterior. De esta forma, y tal y como se ha argumentado en dicho capítulo, es posible diferenciar entre dos grandes clases de enfoques:

- *Enfoques centrados en los datos.* Tal y como ya se ha indicado, tales enfoques consideran los documentos XML como meras bolsas de datos. Para dar soporte a este enfoque se han desarrollado, desde tecnologías que permiten abordar tareas muy especificas de procesamiento tales como XSLT (Birceck et al., 2001), hasta tecnologías de propósito más general como los marcos de procesamiento XML de uso más generalizado, SAX, DOM, StAX etc. (Lam et al., 2008), que permiten abordar tareas más complejas de procesamiento. Tales tecnologías establecen convenios apropiados para la representación interna de los documentos XML (árboles DOM, flujos de eventos SAX, etc.), y definen APIs para agregar la lógica específica de la aplicación (controladores de eventos SAX, iteradores documentales en las propuestas tipo *pull*, etc). En cuanto a la estructura de la lógica de la aplicación, estas tecnologías no imponen ninguna organización específica, más allá de lo que imponga el procesamiento particular a realizar.

- *Enfoques dirigidos por lenguajes.* Frente al planteamiento anterior, en estos enfoques se tiene en cuenta la naturaleza lingüística de los documentos XML. La definición de un tipo de documento XML se corresponde con la definición formal de un lenguaje específico de marcado, y, por tanto, el procesamiento de un documento XML puede abordarse usando técnicas convencionales de procesamiento de lenguajes formales.

Frente a los enfoques centrados en los datos, que no establecen norma alguna sobre la organización de la lógica específica de la aplicación, los enfoques dirigidos por lenguajes permiten dividir el desarrollo de dicha lógica en dos capas bien diferenciadas:

- Una *capa lingüística*, que trata con la naturaleza lingüística del lenguaje de marcado. Dicha capa lingüística puede diseñarse utilizando métodos y técnicas propias de construcción de procesadores de lenguaje.
- Una *capa de lógica específica adicional*, que define el conjunto de operaciones necesario que deben ser invocadas desde la capa lingüística para llevar a cabo el procesamiento en sí.

Esta división no es, de hecho, original de los enfoques dirigidos por lenguajes. Efectivamente, y tal y como se ha discutido en el capítulo anterior, es interesante observar que las lógicas específicas de las aplicaciones en los enfoques orientados a datos se diseñan muchas veces usando métodos genéricos de procesamiento de lenguajes formales (por ejemplo un programa de procesamiento basado en DOM o en StAX es como un traductor recursivo predictivo, o cualquier controlador de contenidos SAX para cualquier lenguaje XML puede ser sistemáticamente diseñado como una máquina de estados con las acciones y elementos de memoria adecuados). Sin embargo, la principal ventaja de los enfoques dirigidos por lenguajes es que permiten especificar la capa lingüística *utilizando definiciones basadas en gramáticas*, a partir de las cuáles es posible generar de forma automática implementaciones eficientes. Dado que dichas especificaciones son de mucho más alto nivel que las codificaciones en lenguajes de propósito general, y que hacen explícita la estructura del lenguaje y la relación de dicha estructura con el procesamiento, el proceso de desarrollo puede verse claramente beneficiado (sobre todo en aquellos casos en los que se involucran lenguajes sofisticados y dotados de estructuras complejas).

En el capítulo anterior se han revisado diversas iniciativas que se ajustan a los enfoques dirigidos por lenguajes. En particular, y dado que las herramientas más usuales para automatizar los procesos de desarrollo durante la construcción de procesadores de lenguaje son los generadores de analizadores sintácticos / traductores, se ha observado la existencia de propuestas tales como RelaxNGCC (Kawaguchi, 2002) o ANTXR (Stanchfield, 2005), que proponen lenguajes de propósito específico para la especificación basada en esquemas de traducción de aplicaciones de procesamiento de documentos XML. Aunque el uso de lenguajes de especificación explícitamente orientados al procesamiento de documentos XML puede facilitar el proceso de especificación, el enfoque también tiene desventajas:

- Por una parte, RelaxNGCC utiliza gramáticas documentales como base al procesamiento. De esta forma, el enfoque incita a reutilizar la misma gramática utilizada en la definición del lenguaje como base a los distintos procesamientos. Esto será adecuado en unos casos, pero no adecuado en otros,

ya que cierto tipo de estructuras son más apropiadas para soportar ciertos tipos de procesamiento que otras. Así mismo, y aun teniendo en cuenta que es posible utilizar gramáticas alternativas equivalentes para cada tipo de procesamiento, el enfoque sigue estando limitado por la propia naturaleza del lenguaje de esquema documental, en el que la única manera de describir la estructura de los contenidos es mediante expresiones regulares. De esta forma, y tal y como se ha discutido en el capítulo anterior, las expresiones regulares no tienen por qué ser necesariamente apropiadas a la hora de guiar procesamientos dirigidos por sintaxis, que se formulan de manera más natural sobre especificaciones basadas en reglas de producción convencionales.

- Por otra parte, aunque ANTXR no se basa, en sí, en ningún lenguaje de esquema, adolece de las mismas limitaciones argumentadas para RelaxNGCC. Efectivamente, ANTXR identifica tipos de elementos con no terminales, lo que no tiene por qué resultar adecuado en todos los casos. En particular, no lo es para lenguajes con modelos de contenidos con un alto grado de estructura implícita, como los ilustrados en (Sarasa et al., 2012b).

A fin de evitar tales desventajas, y, al mismo tiempo, preservar en la medida de lo posible la facilidad de uso ofrecida por estos enfoques dirigidos por lenguajes y apoyados en esquemas de traducción, en esta Tesis se plantea utilizar herramientas de generación de traductores convencionales, en lugar de herramientas más específicas, y combinar las mismas con marcos de procesamiento convencionales. Este hecho conduce a la formulación del primer objetivo de esta Tesis:

> *Objetivo 1. Formular y validar un método general de desarrollo orientado a lenguajes de componentes de procesamiento XML basado en combinar directamente generadores de analizadores sintácticos / traductores* (e.g., JavaCC, CUP) *con marcos de procesamiento* XML *convencionales orientados a eventos* (e.g., SAX, StAX).

3.2.2 Utilización de herramientas de especificación semántica de procesadores de lenguaje para describir e implementar las tareas de procesamiento sobre documentos XML

Tal y como ha evidenciado la revisión realizada en el capítulo anterior, existen diversas propuestas al procesamiento de documentos XML basadas en el formalismo de las gramáticas de atributos. Dichas propuestas ofrecen mecanismos de muy alto nivel para la especificación de tareas de procesamiento, y de un carácter más declarativo que las propuestas basadas en esquemas de traducción (en particular, en una gramática de atributos el desarrollador se ve liberado de considerar el orden explícito en el que las acciones deben ser ejecutadas, ya que dicho orden queda fijado por el proceso de evaluación semántica del formalismo). No obstante, las propuestas analizadas presentan también diversas desventajas, que deben ser consideradas de cara a fomentar su aplicación práctica. Entre ellas, cabe destacar las dos siguientes:

- Un fuerte acoplamiento con la naturaleza EBNF de los formalismos de las gramáticas documentales. Efectivamente, y aunque, como se ha mostrado en el capítulo anterior, existen propuestas de extensión del formalismo de las

gramáticas de atributos a gramáticas EBNF, las propuestas resultantes son menos naturales que las ofrecidas por el formalismo básico. De esta manera, las propuestas al procesamiento XML que se adhieren a esta línea de actuación, y que proponen extensiones de EBNF para el procesamiento XML (Neven, 1999) (Neven, 2005) (Koch et al., 2007) conducen a métodos de especificación bastante antinaturales, en comparación con el formalismo básico de las gramáticas de atributos, hecho que puede comprobarse incluso en los sencillos ejemplos desarrollados en el capítulo anterior. Por otra parte, el desacoplamiento entre sintaxis y semántica propuesto en enfoques como (Psaila et al., 1999) (Havasi, 2002) se aparta de la naturaleza dirigida por sintaxis intrínseca a las gramáticas de atributos, que, precisamente, se apoya en el acoplamiento de los dos aspectos indicados. Mientras que las propuestas de (Gançarski et al., 2002), (Gançarski et al., 2006) de formular el procesamiento sobre gramáticas BNF que resultan de la transformación sistemática de las gramáticas EBNF subyacentes a las gramáticas documentales son fieles al modelo básico de las gramáticas de atributos, también pre-establecen criterios universales para representar la estructura implícita en las expresiones regulares (de nuevo, dicha representación puede ser válida en algunos casos, e inválida en otros). Finalmente, el intento de abordar el problema mediante la transformación de árboles documentales a codificaciones canónicas acotadas en número de hijos adolece de un problema similar: las estructuras resultantes no tienen por qué ser las más adecuadas para todo tipo de procesamiento posible.

- Un fuerte sesgo hacia tipos específicos de procesamiento. Esto es especialmente cierto para propuestas como las de (Neven, 1999) (Neven, 2005) (Koch et al., 2007). Efectivamente, aunque en el capítulo anterior se ha hecho un esfuerzo considerable en extraer la esencia del modelo de procesamiento de dichas propuestas, y en ilustrar cómo los modelos resultantes pueden aplicarse a escenarios más generales, los citados trabajos se restringen al dominio de consultas en documentos XML.

De esta forma, en esta Tesis planteamos abordar dichas desventajas desde la hipótesis básica, mil veces contrastada en el desarrollo de procesadores para lenguajes informáticos más convencionales, de que *el diseño de gramáticas adecuadas es un elemento indispensable en el desarrollo del procesador*. Efectivamente, la gramática que se utiliza en la implementación de un lenguaje no tiene por qué coincidir con la gramática que se utiliza para su definición, sino que tal gramática se adaptará a las necesidades específicas de procesamiento (Klint et al., 2005). Por tanto, en esta Tesis se propone hacer evolucionar la propuesta anterior de uso de herramientas clásicas de procesadores de lenguaje a especificaciones de más alto nivel, basadas en gramáticas de atributos, que puedan ser automáticamente transformadas a las implementaciones basadas en esquemas de traducción. Esto lleva a formular el segundo objetivo de esta Tesis:

> *Objetivo 2. Formular un método general que utilice gramáticas de atributos para especificar declarativamente tareas de procesamiento de documentos XML, y que permita aprovechar estas especificaciones de alto nivel para generar las aplicaciones que implementan el procesamiento descrito.*

3.3 Planteamiento del Trabajo

Partiendo de las consideraciones anteriores y de los objetivos propuestos, podemos resumir que el propósito de esta Tesis es formular un nuevo enfoque lingüístico para el desarrollo de aplicaciones que procesan documentos XML basado en el uso de herramientas de construcción de procesadores de lenguajes, aprovechando de esta forma la naturaleza lingüística que poseen los lenguajes de marcado.

De los objetivos planteados en la sección 3.1 se derivan una serie de requisitos específicos, tanto desde el punto de vista metodológico como desde el punto de vista técnico, que son el punto de partida para enunciar el núcleo de este trabajo de Tesis, y que cristalizan, finalmente, en la implementación de en un entorno de desarrollo denominado XLOP (XML *Language Oriented Processing*). A continuación se desarrollan tales requisitos.

3.3.1 Formulación de un enfoque lingüístico para el desarrollo de aplicaciones de procesamiento de documentos XML basado en herramientas convencionales de construcción de procesadores de lenguaje

3.3.1.1 Formulación del método general

El primer aspecto a abordar en la consecución del objetivo 1, método general de desarrollo basado en la combinación de herramientas convencionales de construcción de procesadores de lenguaje con marcos de procesamiento XML estándar, es definir los distintos aspectos del método, de forma que, en la medida de lo posible, resulten independientes de herramientas y marcos concretos. No obstante, debido a la variabilidad existente en ambos campos (las herramientas de construcción de procesadores, y los marcos de procesamiento XML), en la formulación del método se adoptarán los siguientes sesgos:

- Se considerarán herramientas dirigidas por esquemas de traducción, del tipo de las introducidas en el capítulo anterior. Dado que, tal y como se ha discutido en el capítulo anterior, estas herramientas son las más usuales en el dominio de desarrollo de procesadores de lenguaje, este sesgo no supone, en la práctica, restar demasiada generalidad al enfoque.
- Se considerarán traductores que operan en una única pasada sobre las cadenas de entrada. Tal y como se ha discutido en el capítulo anterior, tales traductores son los más habituales en el dominio, por lo cuál este sesgo tampoco supone una pérdida de generalidad significativa.
- Se considerarán la traducción descendente y la traducción ascendente como modelos de traducción básica. De nuevo, debido a la generalidad y aplicabilidad práctica de dichos modelos, el sesgo no supone pérdida significativa de generalidad.
- Se considerarán traductores que funcionan en modo *push*. De esta forma, tales traductores controlarán el proceso de traducción, requiriendo tokens al analizador léxico conforme lo consideren necesario. Dado que éste es el modelo de organización de traductores clásicos, el enfoque seguirá siendo aplicable a un gran número de escenarios.

- Las gramáticas incontextuales consideradas deberán ser gramáticas no ambiguas. Con ello, se trata de garantizar que la propuesta conduzca a implementaciones lo más eficientes posible, sacrificando la facilidad de uso aportada por herramientas y/o enfoques como (Brand et al., 2001), que no restringen el tipo de gramáticas aceptado.
- Se considerará la existencia explícita de un analizador léxico, así como un acoplamiento mínimo entre el traductor y dicho analizador. En concreto, no se considerará una interacción significativa entre el analizador sintáctico y el léxico, más allá de que el analizador sintáctico solicite *tokens* al léxico, aunque este sesgo no permita aprovechar la potencialidad de enfoques como (Vyk et al, 2006),
- El enfoque se centrará en marcos de procesamiento XML orientados a eventos (tales como SAX o StAX), ya que dicha elección es consistente con la consideración de traductores que operan en una pasada.

Teniendo en cuenta todas estas consideraciones, se analizarán los distintos aspectos del método a seguir, y tal análisis se utilizará en la formulación del método.

3.3.1.2 Validación del método

El método de desarrollo formulado deberá haber sido adecuadamente validado. Para ello, se propone su validación en dos escenarios diferentes:

- Combinación de la herramienta JavaCC con el marco de procesamiento SAX.
- Combinación de la herramienta CUP con StAX.

Tales escenarios son suficientemente representativos, tanto del tipo de herramientas más usuales (generadores de traductores descendentes, en el caso de JavaCC, y generadores de traductores ascendentes, en el caso de CUP), como del tipo de marcos orientados a eventos (marcos *push*, en el caso de SAX, y marcos *pull* en el caso de StAX). Así mismo, en cada uno de los casos se asegurará que la combinación no dependa críticamente de los dos componentes utilizados. Así, por ejemplo, se asegurará que la combinación JavaCC – SAX no dependa de la naturaleza descendente de los traductores JavaCC, ni tampoco de la naturaleza *push* de SAX. Igualmente, en el caso CUP – StAX, se asegurará que dicha combinación no dependa críticamente ni del estilo ascendente de traducción, ni del estilo *pull* de procesamiento XML.

3.3.1.3 Aplicación a casos de estudio

A fin de estudiar la aplicabilidad de la propuesta en la práctica, ésta se aplicará a casos de estudio de complejidad no trivial. En particular, dado que la propuesta es una continuación natural de los trabajos descritos en (Sierra et al., 2004) (Sierra et al., 2006b), se utilizará el principal caso de estudio utilizado en dichos trabajos (generación de aplicaciones de búsqueda de caminos mínimos en redes de metro) como campo de aplicación.

3.3.2 Utilización de herramientas de especificación semántica de procesadores de lenguaje para describir e implementar las tareas de procesamiento sobre documentos XML

3.3.2.1 Formulación del método general

Al igual que ocurre con el objetivo 1, la consecución del objetivo 2, utilización de gramáticas de atributos, radica primeramente en la formulación de una propuesta genérica que norme cómo usar gramáticas de atributos en el desarrollo de aplicaciones de procesamiento XML. En base a las consideraciones realizadas anteriormente, dicha propuesta:

- Utilizará gramáticas incontextuales específicamente diseñadas para cada tarea de procesamiento particular como un componente crítico del proceso de desarrollo. Efectivamente, y al igual que con el uso de esquemas de traducción, las gramáticas incontextuales son el elemento central en el que apoyar el procesamiento. Por tanto, una hipótesis básica en esta Tesis es la necesidad de proporcionar gramáticas específicamente adaptadas a cada tarea, en lugar de tratar de acoplar el procesamiento a gramáticas documentales, cuyo rol principal es definir el lenguaje, no tornarse en un componente básico de apoyo al procesamiento.
- Ofrecerá mecanismos de modularidad básicos que faciliten la aplicación práctica del enfoque. Efectivamente, uno de los principales problemas de los que adolecen las gramáticas de atributos es que, conforme crece la complejidad del lenguaje y del problema de procesamiento, la existencia de una única gramática monolítica puede tornarse en inmanejable (Paakki, 1995). De esta forma, se necesitan mecanismos que permitan descomponer las especificaciones complejas en fragmentos más simples. Afortunadamente, la modularidad intrínseca a las gramáticas de atributos puede ayudar en la formulación de un mecanismo de modularización sencillo, que permita descomponer gramáticas en subgramáticas conforme se descompone la tarea de procesamiento.

Tales consideraciones se tendrán en cuenta en la formulación del enfoque genérico de utilización de gramáticas de atributos como artefactos centrales en el desarrollo de aplicaciones XML.

3.3.2.2 Validación del método

Una vez formulado el método, es necesario proceder a su validación. Para tal fin, se toma el trabajo desarrollado en la consecución del objetivo 1 como base, a fin de refinar el enfoque en uno adecuado para la implementación del método, conforme al cuál:

- Se definirá un lenguaje de especificación sencillo para la especificación de tareas de procesamiento basado en gramáticas de atributos. Dicho lenguaje se adherirá al método general introducido, promoviendo el uso de gramáticas específicamente diseñadas para cada tarea de procesamiento, así como introduciendo mecanismos básicos de modularidad que permitan descomponer gramáticas complejas en fragmentos más manejables.

- Se construirá un generador para dicho lenguaje que permita generar automáticamente implementaciones eficientes en términos de esquemas de traducción.

De esta forma, la conexión entre este objetivo y el anterior se hace evidente: la propuesta de uso de gramáticas de atributos no es más que una forma más fácil de desarrollo de aplicaciones de procesamiento XML mediante herramientas de construcción de procesadores de lenguaje convencionales. De hecho, el citado generador, que se denominará, como ya se ha indicado, XLOP, podrá concebirse como un preprocesador para la combinación entre CUP y un marco orientado a eventos (SAX o StAX).

3.3.2.3 Aplicación a casos de estudio

Al igual que en relación con el objetivo 1, la propuesta basada en gramáticas de atributos se validará sobre casos de estudio no triviales. En particular, dado que esta propuesta es una mejora de la descrita en (Sierra et al., 2008a), se utilizará el caso de estudio relativo a tutoriales socráticos seguido en dicho trabajo como campo en el que probar la aplicabilidad de la propuesta. Así mismo, se utilizará también la comprobación de restricciones sobre documentos de metadatos de objetos de aprendizaje ya aludida anteriormente como caso de estudio adicional.

3.4 A Modo de Conclusión

A fin de promover la realización práctica del enfoque lingüístico al desarrollo de aplicaciones de procesamiento XML dirigido por lenguajes, en esta Tesis se han planteado dos objetivos principales:

- Formulación de un método de desarrollo de aplicaciones de procesamiento XML combinando herramientas clásicas de construcción de procesadores de lenguaje con marcos de procesamiento XML.
- Mejora de dicho método utilizando gramáticas de atributos.

Así mismo, para lograr cada uno de estos objetivos se ha propuesto un método de trabajo consistente en formular enfoques generales, realizar dichos enfoques en términos de marcos de trabajo y herramientas, y aplicar los productos resultantes a casos de estudio realistas.

De esta forma, el siguiente capítulo aporta una discusión integradora del contenido de los artículos que dan soporte a la presente Tesis. Esta discusión relaciona los contenidos de dichas publicaciones con la declaración de objetivos presentada en este capítulo. Por otro lado, el Capítulo 5 incluye una discusión en la que se analiza el grado de cumplimiento de estos objetivos así como las principales líneas de trabajo futuro.

Capítulo 4: Discusión de las Contribuciones de los Artículos

Este capítulo describe las publicaciones editadas en las que se presentan los principales resultados de investigación obtenidos en la consecución de los objetivos en el capítulo anterior. Para ello se sigue, igualmente, el esquema inducido por el planteamiento del trabajo allí esbozado. De esta forma, la sección 4.1 contextualiza las publicaciones en relación con el primero de los objetivos planteados (combinación de herramientas de construcción de procesadores de lenguaje con marcos de procesamiento XML), mientras que la sección 4.2 las contextualiza en relación con el segundo de los objetivos propuestos (uso de gramáticas de atributos como método de especificación de aplicaciones de procesamiento XML).

4.1 Formulación de un enfoque lingüístico para el desarrollo de aplicaciones de procesamiento de documentos XML basado en herramientas convencionales de construcción de procesadores de lenguaje

4.1.1 Formulación del método general

En (Sarasa et al., 2012a) se formula un método general de desarrollo lingüístico de componentes de procesamiento XML dirigidos por la sintaxis que se basa en combinar directamente generadores de analizadores sintácticos / traductores con marcos de propósito general de procesamiento XML orientados a eventos. En este enfoque, la interfaz con los documentos es proporcionada por el marco XML. Dicha interfaz se utiliza para programar un *analizador léxico* XML genérico que transforma documentos en flujos de tokens que serializan la estructura lógica del documento en términos de etiquetas de apertura y de cierre, así como de bloques de texto. A continuación se caracteriza la estructura sintáctica de estos flujos mediante gramáticas incontextuales, a las que se les añade atributos semánticos y acciones semánticas para obtener esquemas de traducción orientados al procesamiento. Estas especificaciones pueden, entonces, utilizarse como entradas

a los generadores a fin de generar automáticamente los componentes de procesamiento XML descritos en dichas especificaciones. El método describe las principales actividades ha realizar, así como su secuenciación en un ciclo de vida que facilita un proceso de producción iterativo/incremental:

- *Configuración del entorno de desarrollo*. Esta actividad consiste en integrar la herramienta de generación de analizadores sintácticos con el entorno de procesamiento XML.
- *Escritura del esquema de traducción orientado al procesamiento.* Durante esta actividad se especifica la tarea de procesamiento mediante un esquema de traducción adecuado. Para ello se escribe, previamente, una gramática incontextual equivalente a la gramática documental del lenguaje de marcado. Seguidamente dicha gramática se anota con acciones semánticas que describen el procesamiento en sí.
- *Implementación de la lógica específica.* Las acciones semánticas que aparecen en el esquema de traducción utilizan servicios y operaciones convencionales, que deben proporcionarse a través de lo que, en el trabajo, se denomina *lógica específica*. Dicha lógica específica puede venir proporcionada, por ejemplo, mediante un conjunto de clases y una *fachada*, que ofrece, en forma de métodos, las citadas operaciones y servicios listos para ser usadas en las acciones semánticas del esquema de traducción.
- *Generación y prueba*. Esta actividad consiste en generar el componente de procesamiento XML aplicando la herramienta de generación de analizadores / traductores al esquema de traducción. El resultado es una implementación efectiva de la capa lingüística de la aplicación en un lenguaje de programación de propósito general, que puede compilarse para producir un componente ejecutable. La aplicación en sí es la combinación de: (i) este componente, (ii) la lógica específica, (iii) el marco de procesamiento XML genérico, y (iv) un programa principal de integración. Dicha aplicación puede, entonces, probarse a fin de resolver posibles defectos o funcionamientos no esperados.

De esta forma, la propuesta recogida en (Sarasa et al., 2012a) satisface este primer aspecto del objetivo 1 planteado en esta Tesis, al formular un método general para desarrollar aplicaciones de procesamiento XML mediante herramientas de construcción de procesadores de lenguaje adecuadamente combinados con marcos de procesamiento XML.

4.1.2 Validación del método

Los resultados relativos a la validación del método se recogen en dos publicaciones diferentes, cada una de ellas orientadas a una de las dos combinaciones ya aludidas en el capítulo anterior:

- En (Sarasa et al., 2008a) se presenta un primer caso de aplicación de la propuesta de combinación, en el que se usa SAX (Simple API for XML) como marco de procesamiento XML y JavaCC (Java Compiler Compiler) como herramienta de generación de analizadores. Dado que SAX es un marco *push*, éste colisiona con el carácter *push* de los traductores generados a partir de JavaCC. Para solucionar el problema se utiliza una solución multihilo, permitiendo que el *analizador léxico* SAX se ejecute en un hilo independiente, y que se sincronice con el traductor a través de la solicitud de *tokens*. Así mismo,

el trabajo desarrolla, además, una arquitectura orientada a la modularización del procesamiento que permite coordinar distintos traductores, cada uno de ellos especificado mediante un esquema JavaCC independiente. Cada traductor en sí corre en su propio hilo, y se sincroniza con el resto a través de una pila semántica común, administrada al estilo de la pila semántica de un traductor descendente predictivo dirigido por tabla.

- En (Sarasa et al., 2009e) se presenta el segundo caso de aplicación del enfoque de combinación. En este nuevo caso de estudio se usa StAX como marco de procesamiento XML y CUP como herramienta de generación de traductores. En este caso la integración es más sencilla, debido al carácter *pull* de StAX, lo que permite integrar directamente analizadores StAX como *scanners* en los parsers generados por CUP. Esto evita el uso de hilos para invertir el control de la ejecución. Por otra parte, aunque este trabajo no permite coordinar múltiples traductores, sí incluye facilidades para la implementación de procesamientos complejos que, en otras condiciones, implicarían múltiples pasadas sobre la estructura de un documento. Para ello, se implementa un estilo de computación conducida por eventos para la programación de las acciones semánticas de los esquemas de traducción, permitiendo incorporar características propias de la evaluación de atributos semánticos dirigida por los datos en gramáticas de atributos.

Por tanto, estos trabajos satisfacen el segundo aspecto planteado en la consecución del objetivo 1, al explorar dos escenarios distintos, y a la vez suficientemente genéricos, de combinación. Así mismo, en ninguno de los casos la naturaleza de uno de los componentes condiciona la del otro componente. Efectivamente, tanto JavaCC como CUP requieren únicamente que los analizadores léxicos sigan una determinada interfaz, de tal forma que la integración con el marco se lleva a cabo implementando dicha interfaz. Por tanto, intercambiar en (Sarasa et al., 2008a) SAX por StAX, o en (Sarasa et al., 2009e) StAX por SAX es una tarea rutinaria. De hecho, en (Sarasa et al., 2009a) se utiliza una combinación de CUP con SAX, en lugar de con StAX, de forma totalmente transparente.

4.1.3 Aplicación a casos de estudio

En (Sarasa et al., 2012a) se describe la aplicación del enfoque al desarrollo de un generador para aplicaciones de búsqueda de caminos mínimos en redes de metro. Dicho generador es una refactorización del descrito en (Sierra et al., 2004) (Sierra et al., 2006b). Así mismo, una vez disponible su lógica específica, la implementación de la capa lingüística tanto en JavaCC y en CUP fue una tarea directa (en particular, la implementación CUP fue especialmente directa debido a la mayor generalidad de las gramáticas LALR frente a las LL soportadas por JavaCC). El generador en sí sigue una arquitectura análoga a la descrita en (Sierra et al., 2004) (Sierra et al., 2006b):

- Un *marco específico de aplicación*, que caracteriza las aplicaciones de búsqueda de caminos mínimos como instanciaciones de las clases de dicho marco, junto al ensamblado de los objetos resultantes.
- Un *generador* que, tomando como entrada un documento XML describiendo las principales características de la aplicación, realiza el citado proceso de instanciación y ensamblado.

No obstante, la descripción en términos gramaticales del componente de generación resulta bastante más mantenible que su descripción en términos de marcos de operacionalización embebidos en lenguajes de propósito general, tal y como se propone en (Sierra et al., 2004) (Sierra et al., 2006b), lo que supone una mejora substancial con respecto a anteriores versiones de esta aplicación XML (en este caso, un generador de aplicaciones dirigido por documentos XML). La constatación de este hecho satisface el tercer aspecto del objetivo 1 planteado por esta Tesis, y, por tanto, completa dicho objetivo.

4.2 Utilización de herramientas de especificación semántica de procesadores de lenguaje para describir e implementar las tareas de procesamiento sobre documentos XML

4.2.1 Formulación del método general

La propuesta de desarrollo de aplicaciones de procesamiento XML mediante gramáticas de atributos se desarrolla en (Sarasa et al., 2012b). Dicha propuesta se basa en las siguientes características:

- Uso de *gramáticas incontextuales* para proporcionar una caracterización declarativa precisa de la estructura lógica de los documentos XML que van a ser procesados. Esta caracterización estructural está orientada hacia las tareas de procesamiento, es decir que es complementaria y más detallada en general que la caracterización descriptiva y orientada a la validación que se consigue con una DTD o con otro formalismo típico de esquema documental. Desde una perspectiva conceptual, el uso de gramáticas en el enfoque hace posible producir, para cada documento XML, un árbol de análisis más detallado que los árboles documentales, adecuado para servir como base estructural al procesamiento dirigido por la sintaxis del documento.
- Uso de *gramáticas de atributos* construidas sobre las gramáticas incontextuales citadas anteriormente. La tarea de procesamiento se caracteriza, de esta forma, en términos de los atributos y ecuaciones semánticas de la gramática.
- Modularización de las tareas de procesamiento mediante *fragmentos de gramáticas de atributos*. En vez de abordarse directamente el desarrollo de una gramática de atributos para una tarea de procesamiento compleja, el desarrollo de la misma pueda llevarse a cabo desarrollando fragmentos más simples que descomponen las tareas originales en tareas más sencillas. La gramática final resulta, entonces, de la composición de los fragmentos de gramáticas de atributos obtenidos en este proceso de descomposición. De esta forma, este estilo de especificación permite describir tareas de procesamiento XML complejas de una forma declarativa que está estrechamente unida a la estructura sintáctica del lenguaje de marcado.

El trabajo descrito en (Sarasa et al., 2012b) cubre, por tanto, el primer aspecto del objetivo 2, ya que formula el método de uso de gramáticas de atributos como instrumentos de desarrollo de aplicaciones de procesamiento de documentos XML

mediante especificaciones de alto nivel, a partir de las cuáles es posible generar automáticamente implementaciones finales para dichas aplicaciones.

4.2.2 Validación del método

4.2.2.1 El sistema XLOP

En (Sarasa et al., 2009a) se describe un entorno de desarrollo de aplicaciones de procesamiento XML basado en gramáticas de atributos denominado XLOP (XML *Language Oriented Processing*). Los principales elementos de dicho entorno son los siguientes:

- *Lenguaje de especificación*. Es un metalenguaje que permite describir las gramáticas de atributos que especifican los procesamientos a realizar sobre los distintos tipos de documentos.
- *Generador*. Este componente toma como entradas gramáticas escritas en el lenguaje de especificación, y produce como salida implementaciones de dichas gramáticas como esquemas de traducción CUP. Para llevar a cabo la implementación de atributos utiliza una estrategia de evaluación de atributos dirigida por los datos, análoga a la introducida en (Sarasa et al., 2009e).
- *Entorno de ejecución*. Este componente incluye la maquinaría básica necesaria para llevar a cabo la evaluación semántica, así como la integración de los traductores generados con un marco de procesamiento XML. En la versión de XLOP descrita en (Sarasa et al., 2009a), el marco de procesamiento considerado es, como ya se ha indicado anteriormente, SAX.

4.2.2.2 Modelo de desarrollo en XLOP

En (Sarasa et al., 2009d) se muestra cómo XLOP realiza de manera efectiva el enfoque de desarrollo lingüístico de aplicaciones de procesamiento XML. En concreto, dicho trabajo ilustra el método de desarrollo de XLOP, basado en un conjunto de actividades similares a las propuestas en (Sarasa et al., 2012a) y descritas anteriormente, y también implícitas en el enfoque basado en gramáticas de atributos de (Sarasa et al., 2012b). Efectivamente, el desarrollo de aplicaciones con XLOP involucra las siguientes actividades:

- *Escritura de la gramática incontextual.* Los desarrolladores escriben la gramática incontextual para el tipo de documentos y procesamiento a desarrollar, y la codifican en términos del lenguaje de especificación de XLOP.
- *Especificación del procesamiento con atributos y ecuaciones semánticas.* A partir de la gramática de la actividad anterior y usando nuevamente el lenguaje de especificación, se describe la tarea de procesamiento que se quiere llevar a cabo con los documentos XML mediante una gramática de atributos, siendo necesario especificar los atributos semánticos de cada símbolo no terminal y las ecuaciones semánticas que permiten calcular los valores de los atributos.
- *Implementación de las funciones semánticas*. Esta actividad completa la caracterización del procesamiento mediante la implementación de las funciones semánticas que se usan en las ecuaciones semánticas de la gramática de atributos. El lenguaje de especificación de XLOP está

estrechamente acoplado con Java, de manera que las funciones semánticas que se usan en las ecuaciones de la gramática de atributos para calcular los valores de las instancias de atributos, deben ser implementadas por los desarrolladores como métodos de una clase java específica de la aplicación denominada *clase semántica*. Esta clase actúa como interfaz con el resto del software específico de la aplicación, que puede ser provisto mediante un conjunto de clases java.

- *Generación*. La gramática de atributos es procesada por un generador, el cual produce automáticamente el componente/aplicación de procesamiento XML.
- *Prueba*. Los desarrolladores prueban la aplicación generada sobre diferentes documentos XML. En caso de existir algún defecto, se retorna a las actividades previas para corregirlo.

Obsérvese que, de esta forma, XLOP puede concebirse como un preprocesador que permite generar automáticamente implementaciones de gramáticas de atributos basadas en esquemas de traducción que, al estilo de las abordadas en la consecución del objetivo 1, pueden ser transformadas en aplicaciones finales mediante herramientas de generadores de analizadores sintácticos / traductores (en este caso, mediante CUP). Este hecho justifica la analogía entre el modelo de desarrollo de XLOP y el descrito en (Sarasa et al., 2012a) para la combinación de esquemas de traducción y marcos de procesamiento XML.

4.2.2.3 Optimizaciones en XLOP

Una vez disponible la versión inicial de XLOP descrita en (Sarasa et al., 2009a), durante el desarrollo de esta Tesis se ha trabajado extensivamente en su mejora y optimización, a fin de facilitar, por una parte, su uso durante el desarrollo de aplicaciones, y permitir, por otra, la generación de aplicaciones eficientes. De esta forma:

- Se ha dotado a la herramienta de mecanismos de modularización mediante fragmentos de gramáticas de atributos, tal y como se describe en (Sarasa et al., 2012b).
- Así mismo, se ha dotado a la herramienta de capacidades de análisis de las especificaciones, a fin de realizar diversas optimizaciones sobre los esquemas de traducción generados.

En particular, las mejoras relativas a la realización de optimizaciones durante el proceso de generación de esquemas de traducción son especialmente relevantes, ya que permiten generar aplicaciones capaces de procesar documentos con un nivel aceptable de eficiencia. Las optimizaciones, cuyos detalles se describen en (Sarasa et al., 2012b) se basan en los siguientes procesos:

- Análisis de los autómatas LR(0) de las gramáticas incontextuales subyacentes a las especificaciones, a fin de añadir marcadores a dichas gramáticas. Tal y como se ha indicado en el capítulo 2, dichos marcadores permiten, por una parte, alojar atributos heredados de otros no terminales, y, por otra, simular la ejecución de acciones en el interior de los cuerpos de las reglas durante el análisis ascendente.
- Alojamiento de atributos heredados en marcadores. Utilizando la información proporcionada por el proceso de añadido de marcadores, XLOP puede

determinar conjuntos de atributos heredados alojables en dichos marcadores. Este hecho, combinado con el mecanismo de evaluación dirigido por los datos, permite optimizar la memoria adicional requerida por el proceso de evaluación semántica, al permitir adelantar dicha evaluación, e intercalarla con el análisis.

– Determinación de atributos evaluables durante el análisis. Este proceso permite evaluar atributos *al vuelo*, evitando la creación de las estructuras adicionales requeridas por la evaluación dirigida por los datos.

Aparte de estas optimizaciones, tal y como se describe en (Sarasa et al., 2012b), en la versión actual de XLOP se ha mejorado el mecanismo de evaluación dirigida por los datos, y se ha optimizado también algunos aspectos del algoritmo de análisis sintáctico incluido en el entorno de ejecución de CUP a fin de mejorar el rendimiento (incluyendo, en concreto, un mecanismo de reciclaje de los objetos intermedios creados, a fin de disminuir el uso del *heap*). De esta forma, como se ilustra en (Sarasa et al., 2012b), los rendimientos en tiempo y en memoria de las aplicaciones producidas son comparables a las obtenidas manualmente mediante el uso de marcos dirigidos por eventos (SAX, StAX), y bastante mejores que los construidos con DOM. Por su parte, como se ilustra también en (Sarasa et al., 2012b), XLOP es mucho más expresivo y usable que la codificación manual mediante marcos de procesamiento de propósito general, sobre todo en aquellos casos que involucran lenguajes complejos, dotados de estructuras sofisticadas, o con un alto grado de estructura implícita en los modelos de contenidos de los tipos de elementos.

4.2.2.4 Especializaciones de XLOP

Por último, en esta Tesis también se ha explorado la posibilidad de llevar a cabo especializaciones adicionales en el lenguaje de especificación XLOP para permitir especificaciones más naturales en determinados dominios. En particular, en (Sarasa et al., 2011) se muestra cómo utilizar XLOP como una herramienta de implementación de servicios web XML tipo REST (Fielding, 2000), (Richardson et al., 2007). Dado que dichos servicios producen como salida documentos XML, se introdujo en el lenguaje azúcar sintáctico adicional para facilitar la composición de dichos documentos. La extensión en sí se reducía a la notación XLOP básica traduciendo la notación de composición de documentos en invocaciones al API DOM.

4.2.3 Aplicación a casos de estudio

En (Sarasa et al., 2009c) y en (Sarasa et al., 2009d) se ejemplifica el uso de XLOP en el caso de estudio de <e-Tutor>, el ya citado marco para el desarrollo de tutores socráticos interactivos basado en los trabajos de Alfred Bork (Bork, 1985), y utilizado en (Sierra et al., 2008a) como caso de estudio en un enfoque precursor al desarrollado en esta Tesis. <e-Tutor> interpreta descripciones de *tutoriales* en los cuáles el estudiante se somete a *problemas*, para los que construye *soluciones* a través de un diálogo *socrático* (*maestro – discípulo*). El contenido de aprendizaje se presenta con preguntas interactivas y con respuestas que dependen de las interacciones que el usuario va realizando conforme avanza el tutorial, de manera que cuando se elige una respuesta se proporciona una realimentación apropiada y

se decide el próximo paso a llevar a cabo en el proceso de aprendizaje. Para implementarlo se usa un mecanismo simple basado en contadores que se asocian a cada posible respuesta de cada pregunta, de manera que cada vez que el estudiante da una respuesta, se incrementa el contador asociado. Así la realimentación y el siguiente paso a dar dependen del valor de dichos contadores. La característica fundamental de <e-Tutor> es la posibilidad de describir un tutorial en un documento XML, de manera que el procesamiento del mismo permite su construcción. En este sentido, los trabajos descritos en (Sarasa et al., 2009c) y en (Sarasa et al., 2009d) ilustran el uso de XLOP para construir un procesador optimizado para este tipo de documentos XML.

Por su parte, en (Sarasa et al., 2009b), (Sarasa et al., 2009c), (Sarasa et al., 2011) y (Sarasa et al., 2012b) se utiliza *Chasqui* como escenario de ejemplificación. *Chasqui* es una plataforma e-Learning utilizada en la virtualización del patrimonio, que permite la creación y uso educativo de repositorios de objetos de aprendizaje. Los objetos de aprendizaje en *Chasqui* tienen asociados documentos XML que describen una secuencia de ítems de metadatos organizados jerárquicamente, de manera que cada ítem de metadatos posee un nombre y un valor. Los ítems no están predeterminados a priori, sino que se crean de manera colaborativa, conforme se añaden nuevos objetos al repositorio. Así mismo, en *Chasqui* es posible imponer restricciones sobre los documentos XML asociados con los objetos que se añaden al repositorio. Estas restricciones se formulan usando un repertorio completo de asertos básicos, que pueden combinarse utilizando los operadores booleanos habituales (`and`, `or` y `not`), y que se representan internamente utilizando un lenguaje de marcado XML. La tarea de procesamiento consiste, entonces, en comprobar el cumplimiento o incumplimiento de las restricciones sobre los documentos de metadatados. De esta forma:

- En (Sarasa et al., 2009b) y (Sarasa et al., 2009c) se considera un lenguaje de descripción de restricciones basado en la codificación de la sintaxis abstracta de las restricciones en XML.
- En (Sarasa et al., 2011) ilustra otro experimento realizado, esta vez con un lenguaje de restricciones más simple, consistente únicamente en secuencias de asertos básicos.
- Por último, en (Sarasa et al., 2012b) se utiliza una codificación XML más simple de las restricciones, pero que ofrece muchas más complejidades de cara al procesamiento, y se muestra cómo el uso del enfoque basado en gramáticas de atributos propuesto puede tratar con dicha codificación de manera análoga al resto de los casos, sin nuevas dificultades añadidas.

De esta forma, la aplicación práctica de XLOP a estos casos de estudio finaliza la consecución del objetivo 2 planteado en esta Tesis.

Capítulo 5: Conclusiones y Trabajo Futuro

Los capítulos anteriores han mostrado una nueva perspectiva para el desarrollo de aplicaciones de procesamiento de documentos XML desde el ámbito de la construcción de procesadores de lenguajes, aprovechando de esta manera la naturaleza lingüística de los lenguajes de marcado. En este sentido se ha mostrado que es posible definir un modelo de desarrollo y herramientas que soporten este nuevo enfoque. Tal y como se ha discutido en el capítulo anterior, estos aspectos se detallan en las distintas publicaciones que integran esta memoria de Tesis. De esta forma, con el presente capítulo se concluye esta memoria, resumiendo las principales aportaciones (sección 5.1), y describiendo algunas líneas de investigación futura (sección 5.2)

5.1 Principales Aportaciones

En esta sección se enuncian las principales aportaciones realizadas en esta Tesis. Más concretamente, y en base a la discusión mantenida en los capítulos precedentes, pueden destacarse las siguientes:

- Propuesta de un enfoque para el desarrollo de aplicaciones de procesamiento de documentos XML basado en la combinación de herramientas de construcción de procesadores de lenguaje y marcos de procesamiento XML estándar.
- Implementación de dos entornos para el desarrollo de aplicaciones de procesamiento XML basados en dicho enfoque.
- Refinamiento del enfoque para la utilización de las gramáticas de atributos como formalismo para especificar las tareas de procesamiento sobre documentos XML.
- Implementación de un entorno de desarrollo basado en dicho refinamiento.
- Uso práctico de las propuestas de desarrollo lingüístico de aplicaciones XML.

Los siguientes puntos presentan con más detalle cada una de estas aportaciones.

5.1.1 Enfoque al desarrollo de aplicaciones de procesamiento XML mediante la combinación de herramientas de construcción de procesadores de lenguaje y marcos de procesamiento XML

En este trabajo de Tesis se ha presentado un nuevo enfoque lingüístico para el desarrollo de aplicaciones de procesamiento de documentos XML que se basa en aprovechar la naturaleza lingüística de los lenguajes de marcado, y por tanto la estructura de la información que poseen los documentos XML. Para ello se propone combinar directamente herramientas de generación de analizadores sintácticos / traductores convencionales con marcos de procesamiento XML estándar orientados a eventos. Esta combinación define un proceso de desarrollo que se caracteriza por:

- Promover la especificación a alto nivel de las capas lingüísticas de las aplicaciones de procesamiento mediante esquemas de traducción.
- Promover un enfoque generativo, según el cuál es posible generar la capa lingüística así especificada mediante una herramienta de generación de traductores adecuada. Los traductores generados interactúan con el marco genérico de procesamiento XML mediante un *analizador léxico* XML configurable que permite mapear a flujos de tokens los flujos de ítems de información obtenidos al procesar los documentos XML.

Algunas de las ventajas de este enfoque sobre otros enfoques más convencionales, dirigidos por los datos, son:

- La *eficiencia* del proceso de desarrollo al utilizar herramientas que generan las aplicaciones de manera semiautomática a partir de especificaciones formales de alto nivel, permitiendo, de esta forma, disponer de entornos de desarrollo más declarativos, sistemáticos y mantenibles.
- La *facilidad para especificar* tareas de procesamiento complejas sobre lenguajes con estructuras sofisticadas, al brindar un mecanismo declarativo de descripción de la estructura, así como formas de llevar a cabo procesamiento dirigido por dichas estructuras.
- La *facilidad para mantener las aplicaciones* desarrolladas, dado que el procesamiento se describe en las especificaciones de alto nivel.
- El *carácter genérico* del método, dado que es aplicable al desarrollo de cualquier aplicación de procesamiento.

Por su parte, frente a propuestas similares que utilizan esquemas de traducción en el desarrollo de aplicaciones de procesamiento XML, la propuesta realizada en esta Tesis ofrece como principal aportación el no comprometerse con un lenguaje de especificación particular, que introduzca, a su vez, sesgos en relación a cómo deben interpretarse las estructuras sintácticas implícitas en los modelos de contenido. De esta forma, el enfoque promueve el uso de gramáticas incontextuales adecuadas para cada lenguaje y tarea de procesamiento. Estas gramáticas describen el lenguaje de forma que resulte adecuada al procesamiento, permitiendo asociar con cada documento XML correcto árboles de análisis detallados y adecuados para llevar a cabo el procesamiento dirigido por la sintaxis. De esta forma, el propósito de las mismas es claramente diferente al de las gramáticas documentales, orientadas a definir el lenguaje en lugar de a servir como soporte a posteriores procesamientos.

5.1.2 Implementación de entornos para el enfoque al desarrollo de aplicaciones de procesamiento XML mediante la combinación de herramientas de construcción de procesadores de lenguaje y marcos de procesamiento XML

El trabajo de Tesis propone dos entornos concretos que muestran la factibilidad práctica de combinar herramientas de generación de analizadores sintácticos / traductores con marcos de procesamiento XML orientados a eventos:

- Por una parte, se ha propuesto un entorno basado en la combinación de JavaCC y de SAX. Dicho entorno incluye, así mismo, soporte para especificaciones modulares, permitiendo especificar y coordinar diferentes aspectos mediante múltiples esquemas de traducción.
- Por otra parte, se ha propuesto otro entorno que combina CUP y StAX. Dicho entorno incluye soporte para tratar con cómputos dirigidos por dependencias, a fin de capturar problemas que, en general, requerirían de más de una pasada sobre el documento.

Aparte de servir como banco de pruebas para justificar la viabilidad práctica de la propuesta de desarrollo, tales entornos constituyen aportaciones importantes desde el punto de vista práctico / aplicado, ya que, al basarse en herramientas y soluciones ampliamente extendidas, permiten una aplicación práctica inmediata de la propuesta a escenarios de desarrollo reales.

5.1.3 Propuesta de utilización de las gramáticas de atributos como formalismo para especificar las tareas de procesamiento sobre documentos XML

Otro resultado importante de esta Tesis se refiere a la propuesta de utilizar las gramáticas de atributos como medio de especificación práctico de tareas de procesamiento de documentos XML. Las gramáticas de atributos proporcionan un formalismo declarativo de mayor nivel para la descripción de aplicaciones de procesamiento XML que los marcos de procesamiento de propósito general o los esquemas de traducción. Efectivamente, en una gramática de atributos, el orden de las producciones y las reglas semánticas no altera el significado de la gramática de atributos, de manera que estas gramáticas pueden ser enriquecidas añadiendo nuevos atributos y nuevas reglas semánticas, de forma que el entorno de evaluación reestructura de manera automática el proceso y el orden de evaluación. Como consecuencia, el formalismo proporciona un mecanismo dotado de una naturaleza modular intrínseca, que es especialmente apropiado para especificar procesos que involucran complejos flujos de información sobre estructuras sintácticas sofisticadas.

La propuesta realizada en esta Tesis surge como una maduración directa de las propuestas relativas al uso de esquemas de traducción descritas anteriormente. Efectivamente:

- Por una parte, la propuesta pone en valor la modularidad intrínseca del formalismo permitiendo especificar tareas de procesamiento complejas mediante múltiples fragmentos de gramáticas de atributos que descomponen las tareas originales en subtareas más simples y manejables. El resultado es una forma simple y sistemática de incorporar las características de

modularidad aventuradas por la combinación JavaCC – SAX a la que se ha hecho alusión anteriormente.

– Por otra parte, la propuesta incorpora directamente un modelo de cómputo basado en dependencias entre atributos semánticos, lo que captura directamente el mecanismo de cómputo dirigido por dependencias incorporado en la combinación CUP – StAX anteriormente citada.

Así mismo, frente a otras propuestas similares que utilizan gramáticas de atributos en relación con el procesamiento de XML, esta propuesta hereda también la característica distintiva señalada anteriormente en relación con el uso de esquemas de traducción: formulación de gramáticas incontextuales orientadas a tareas como parte central del proceso de desarrollo, frente a los enfoques de reconciliación de las gramáticas de atributos con la naturaleza EBNF intrínseca de las gramáticas documentales seguidos por otras propuestas (desacoplo entre sintaxis y semántica, transformación predefinida de gramáticas, extensión del formalismo de gramáticas de atributos a gramáticas EBNF, o transformación predeterminada de árboles documentales a árboles acotados).

5.1.4 Entorno para el desarrollo de aplicaciones XML basado en gramáticas de atributos

Como resultado de este trabajo de Tesis también se ha implementado un entorno de programación denominado XLOP (XML *Language-Oriented Processing*) que automatiza y da soporte a la propuesta de uso de gramáticas de atributos citada anteriormente. Efectivamente, XLOP toma como entrada la especificación de una tarea de procesamiento XML en términos de un conjunto de fragmentos de gramáticas de atributo, y produce como salida una implementación de la capa lingüística de la aplicación. Junto con la lógica específica adicional de la aplicación, y junto con el entorno de ejecución de XLOP (que incluye, por una parte, soporte para la evaluación de atributos dirigida por los datos, y, por otra, una conexión con un marco XML orientado a eventos estándar), dicha implementación da lugar a la aplicación de procesamiento XML completa.

XLOP se puede concebir como un preprocesador que eleva, mediante el uso de gramáticas de atributos, el nivel del enfoque basado en esquemas de traducción aludido anteriormente. Efectivamente, el producto finalmente generado por XLOP es un esquema de traducción implementado en CUP, integrable con un marco XML orientado a flujos mediante un analizador léxico XML, configurado también por el propio XLOP como resultado del proceso de generación.

Inicialmente XLOP se implementó realizando una codificación simple del mecanismo de evaluación dirigido por los datos, así como integrando los traductores resultantes con SAX. Posteriormente, y también como parte del trabajo de esta Tesis, dicha implementación se refinó a fin de mejorar la eficiencia de las aplicaciones finales, mostrándose que bajo una elección adecuada de la gramática incontextual, es posible generar aplicaciones con un rendimiento comparable a las codificadas manualmente. Para ello, se incluyó en XLOP una serie de procesos de análisis y optimización de los esquemas de traducción finales, entre los que destaca un método original de implementación de gramáticas de atributos sobre generadores de analizadores LR convencionales, que está basado en la teoría de gramáticas LR-atribuidas (Akker et al., 1990).

5.1.5 Usos prácticos de las propuestas de desarrollo dirigido por lenguajes de aplicaciones XML

Para finalizar, aunque no por ello menos importante, en esta Tesis se ha estudiado la viabilidad práctica de las propuestas realizadas mediante su aplicación práctica a distintos casos de estudio concernientes a aplicaciones intensivas en información y a aplicaciones educativas:

- El generador de aplicaciones de búsqueda de rutas en redes de metro descrito en (Sierra et al., 2004) (Sierra et al., 2006b).
- El generador de tutores socráticos propuesto en (Sierra et al., 2008a).
- La comprobación del cumplimiento o violación de restricciones sobre documentos de metadatos en el sistema de gestión de objetos de aprendizaje *Chasqui*.

Estas experiencias, aún sin constituir una batería de casos de estudio exhaustiva (batería cuyo desarrollo excedería con mucho los objetivos de este trabajo de Tesis), arrojan evidencia positiva acerca de la viabilidad del desarrollo de aplicaciones lingüístico propuesto en esta Tesis.

5.2 Trabajo futuro

Esta sección concluye el capítulo presentando algunas líneas de trabajo futuras de investigación que se desprenden de esta Tesis. Más concretamente, se consideran las siguientes líneas de investigación como las más prometedoras para continuar el trabajo iniciado en esta Tesis:

- Mejora de XLOP.
- Comprobación de la equivalencia entre representaciones gramaticales.
- Avances en la especificación modular de tareas de procesamiento XML.
- Implementación de gramáticas de atributos mediante herramientas de generación de compiladores.
- Aplicación de XLOP y del enfoque dirigido por lenguajes a otros dominios de aplicación.
- Otros usos de gramáticas de atributos como herramientas de construcción de software.
- Mecanismos para facilitar la adopción del enfoque lingüístico entre desarrolladores.

Los siguientes puntos motivan brevemente cada una de estas líneas de investigación y de trabajo futuro.

5.2.1 Mejora de XLOP

En esta línea de trabajo se proponen diversas mejoras sobre el entorno de desarrollo XLOP, que, a su vez, redunden en mejoras en la propuesta basada en gramáticas de atributos en la que éste se apoya. El primer grupo de mejoras se refieren al lenguaje de especificación. Tales mejoras afectan a los siguientes aspectos:

- En las ecuaciones semánticas de las gramáticas de atributos, se pretende dar soporte a funciones no estrictas (funciones que no requieren la evaluación de todos sus parámetros).
- Se planea también incluir un sistema de tipado en las especificaciones admitidas por XLOP, que permita un chequeo estático del tipado, así como la inferencia del tipo de cada atributo.
- También se tiene previsto dar soporte a operadores definidos por el usuario, así como a la definición de restricciones sobre contenidos textuales y atributos de elementos.

Por otra parte, también se prevé la realización de diversas mejoras en el entorno propiamente dicho. En particular se plantean las siguientes mejoras:

- Desarrollo de un modelo de depuración, y un entorno gráfico.
- Desarrollo de un entorno de desarrollo integrado para XLOP.

5.2.2 Comprobación de la equivalencia entre representaciones gramaticales

Uno de los aspectos básicos de las propuestas realizadas en esta Tesis consiste en obtener una gramática incontextual que represente la estructura lógica de los documentos XML a procesar. En general para ello se dispone como punto de partida de una DTD o esquema del lenguaje de marcado que sirve de modelo para los documentos XML. El paso de la DTD o esquema a la gramática independiente del contexto se realiza de manera manual. Actualmente no existe ningún algoritmo ni resultado que permita garantizar la equivalencia de las dos representaciones del lenguaje de marcado (problema que, en el caso más general, resulta indecidible).

En este sentido una línea de investigación abierta consiste en estudiar bajo qué condiciones es posible asegurar que las dos representaciones son equivalentes, y encontrar un algoritmo que automatice el proceso de comprobación de la equivalencia entre las representaciones de lenguajes de marcado en forma de DTDs y en forma de gramáticas independientes del contexto, bajo las limitaciones sobre ambos tipos de gramáticas necesarias para garantizar la decidibilidad del problema. De hecho, ya se dispone de algunos resultados en esta línea, tal y como se describe en (Temprado-Battad et al., 2011).

5.2.3 Avances en la especificación modular de tareas de procesamiento XML

Una de las ventajas del uso de las gramáticas de atributos para la descripción de tareas de procesamiento XML es su carácter modular que facilita que la descripción se pueda realizar mediante un proceso iterativo-incremental dividiendo la tarea original compleja en subtareas que son especificadas individualmente como fragmentos de gramáticas de atributos. Sin embargo en este proceso siempre se toma como base para la especificación la misma gramática incontextual. En algunos procesamientos puede no ser útil utilizar como base de la definición de la gramática de atributos asociada al procesamiento, la misma gramática incontextual.

De esta forma, una línea de investigación abierta consiste en estudiar la posibilidad de utilizar para los diferentes fragmentos de gramáticas de atributos gramática incontextuales potencialmente distintas, y expresamente diseñadas para el aspecto de procesamiento que se quiere realizar. En (Temprado-Battad et al., 2010) aparecen algunos resultados preliminares en esta línea.

Esta línea de investigación constituye el tema central de las Tesis doctoral de Bryan Temprado Battad "Un Enfoque Metalingüístico al Procesamiento de Documentos XML", que actualmente se encuentra en fase de realización.

5.2.4 Implementación de gramáticas de atributos mediante herramientas de generación de compiladores

Uno de los resultados obtenidos en la implementación del entorno de desarrollo de XLOP es el haber definido un mecanismo de implementación de las gramáticas de atributos como esquemas de traducción dirigidos por la sintaxis. Sin embargo este mecanismo está particularizado al contexto de las características de la herramienta de generación de traductores CUP. Sin embargo no se ha investigado si se puede implementar con cualquier tipo de herramienta de generación de traductores, o si el proceso es dependiente de la estrategia de análisis de los traductores generados. Por otra parte, la posibilidad de convertir gramáticas de atributos en esquemas de traducción iniciales, que, posteriormente, puedan refinarse, así como de disponer de vínculos bidireccionales entre esquemas de traducción y gramáticas, es un aspecto cuyo estudio puede ser de máximo interés como mecanismo de implementación de lenguajes.

De esta forma, surge una línea de investigación consistente en tratar de generalizar este resultado y estudiar si se puede definir un método general para implementar gramáticas de atributos mediante herramientas de generación de compiladores. En (Rodríguez-Cerezo et al., 2011b) (Rodríguez-Cerezo et al., 2012b) se describen algunos resultados preliminares en esta línea.

5.2.5 Aplicación de XLOP y del enfoque dirigido por lenguajes a otros dominios de aplicación

Otra línea de trabajo es la aplicación y adaptación de los resultados obtenidos a diferentes dominios de aplicación. En este sentido durante el desarrollo de este trabajo de Tesis se ha aplicado con éxito los resultados de la tesis en el desarrollo de generadores basados en XML, tutoriales socráticos, o la comprobación de restricciones sobre los metadatos de objetos de aprendizaje. Sin embargo los resultados de esta tesis son aplicables a cualquier tipo de aplicación que procese documentos XML. En este sentido hay varios dominios de aplicación industriales que, por su naturaleza, son susceptibles de investigación. Entre ellos cabe destacar los siguientes:

- *Gestión de facturas electrónicas*. El modelo de información de las facturas electrónicas se basa en lenguajes de marcado XML, y de hecho existen diferentes tipos de facturas que responden a diferentes lenguajes de marcado.
- *Desarrollo de editores web*. El desarrollo de editores web, basados en formularios, constituye un campo interesante de aplicación, dado que son aplicaciones altamente configurables, de manera que la información de configuración podría almacenarse en documentos XML cuyo procesamiento generase el editor correspondiente.
- *Aplicación a los repositorios digitales*. Por último las aplicaciones que actúan de repositorios digitales responden a una estructura común que permite configurarlas mediante documentos XML, de manera que la aplicación en sí

puede ser vista como una aplicación XLOP. Este enfoque permitirá crear y mantener de una manera fácil y particularizada este tipo de aplicaciones.

Actualmente los dos últimos de estos campos de aplicación se están explorando en el contexto del proyecto del subprograma de investigación no orientada "Un Enfoque Generativo a la Construcción de Herramientas de Producción y Despliegue de Objetos de Aprendizaje en el Campus Virtual" (TIN2010-21288-C02-01). En particular, en dichos campos se está abordando la siguientes problemática:

- Configuración a partir de documentos XML de la herramienta de anotación de textos literarios digitalizados @note. Esta es una herramienta desarrollada bajo el auspicio de Google (*Digital Humanities Award Programs* 2010, en el que el grupo de investigación ILSA obtuvo, en colaboración con el grupo de investigación LEETHI[1] de la Facultad de Filología de la UCM, una de las 12 ayudas concedidas por Google a nivel mundial para el desarrollo de las Humanidades Digitales). La herramienta en sí permite anotar textos con anotaciones de forma colaborativa. El objetivo de la experiencia es permitir configurar fácilmente dicha aplicación a partir de la descripción de sus principales aspectos en documentos XML externos.
- Configuración a partir de documentos XML de la herramienta de creación de colecciones de objetos digitales OdA-Virtual. OdA-Virtual es, de hecho, la evolución del sistema *Chasqui* para dar lugar a una plataforma independiente de dominio que permita la creación sencilla por parte de expertos del dominio de colecciones de objetos digitales en diferentes escenarios. Al igual que en el caso anterior, en esta experiencia se está diseñando la forma de instanciar colecciones iniciales a partir de documentos.

En cuanto al primero de los campos, el procesamiento de facturas electrónicas, actualmente se mantienen contactos con empresas como BABEL o DyR a fin de plantear una propuesta de proyecto o contrato de investigación que permita soportar adecuadamente las experiencias en el mismo.

5.2.6 Otros usos de las gramáticas de atributos como herramientas de desarrollo de software

Mientras que las gramáticas de atributos han sido tradicionalmente utilizadas como herramientas de diseño e implementación de lenguajes de programación, en esta Tesis se ha mostrado que el formalismo puede ser útil para otros propósitos diferentes (en este caso, procesamiento de un cierto tipo de información semi-estructurada). Este hecho abre la puerta a todo un elenco de líneas de trabajo, todas ellas orientadas a poner en valor las gramáticas de atributos como herramientas de desarrollo de software:

- Utilización de gramáticas de atributos para el procesamiento de otros tipos de información semiestructurada. En concreto, los resultados de esta Tesis pueden extrapolarse de forma más o menos directa al procesamiento de textos JSON, YAML, o incluso expresiones-s como las usadas en LISP.
- Uso de gramáticas de atributos como mecanismos de transformación de modelos. Este aspecto supone utilizar gramáticas de atributos en Ingeniería del Software Dirigida por Modelos (Stahl et al., 2006), como mecanismos de

[1] Literaturas Españolas y Europeas: del Texto al Hipertexto (http://www.ucm.es/info/leethi/)

especificación de transformaciones entre modelos. En (Gracia-Benitez, 2010) se muestra ya algún trabajo preliminar en esta línea.

- Procesamiento de grafos mediante gramáticas de atributos. La transformación de modelos es un caso particular de procesamiento de grafos. Por tanto, los resultados obtenidos en la línea anterior podrán generalizarse a este caso más general, permitiendo, así mismo, procesar cuerpos de información estructurada en forma de grafo, tales como cuerpos RDF o OGDL (notaciones ambas que permiten describir grafos, en lugar de árboles).
- Especificación de la interacción en aplicaciones interactivas y web. La idea de esta línea es retomar enfoques como los descritos en (Nymeyer, 1995) a fin de especificar declarativamente controladores de aplicaciones interactivas y web mediante gramáticas de atributos, y generar automáticamente dichos controladores a partir de dichas especificaciones.

Todas éstas son, de hecho, líneas de investigación que se encuentran actualmente en desarrollo en el seno del grupo ILSA.

5.2.7 Mecanismos para facilitar la adopción del enfoque lingüístico

Por último, una última línea de investigación consiste en introducir mecanismos que faciliten la adopción del enfoque lingüístico como enfoque para llevar a cabo el procesamiento de documentos XML, así como otros aspectos del desarrollo de software. Uno de los aspectos a cubrir por esta línea es el de la correcta preparación de futuros desarrolladores en las habilidades requeridas por estas técnicas. Algunos trabajos preliminares al respecto son (Sierra et al., 2008b), (Rodríguez-Cerezo et al., 2011a), (Rodríguez-Cerezo et al., 2012a), todos ellos orientados a proporcionar herramientas educativas para el aprendizaje del formalismo de las gramáticas de atributos.

Esta línea de investigación es el tema de la Tesis Doctoral de Daniel Rodríguez Cerezo “Herramientas Educativas para Facilitar la Adopción de la Ingeniería de Lenguajes Software entre los Desarrolladores Informáticos”, actualmente en curso de realización.

Capítulo 6: Ártículos Presentados

A continuación se incluyen los artículos editados que se aportan como parte de esta Tesis Doctoral.

6.1 Building a Syntax Directed Processing Environment for XML Documents by Combining SAX and JavaCC

Cita completa:

Sarasa-Cabezuelo, A., Navarro-Iborra, A., Sierra-Rodriguez, J.L, Fernández-Valmayor, A. Building a Syntax Directed Processing Environment for XML Documents by Combining SAX and JavaCC. *Third International Workshop on XML Data Management Tools & Techniques* (XANTEC '08) - *19th International Conference on Database and Expert Systems Application* (DEXA'08), 1-5 de Septiembre de 2008, Turin, Italia. DEXA'08 Workshops. IEEE Computer Society, pp.256-260

Resumen original de la contribución:

In this paper we show how to integrate JavaCC, a popular translator-generation tool, with any standard XML parsing environment supporting the SAX specification. This integration lets developers build efficient XML processing applications which act as left-to-right, one-pass translators. The integration also facilitates the maintenance of these applications, since they are specified as syntax-directed translation schemas instead of being directly programmed in a general-purpose programming language. This integration proposal also allows for exploiting the modularization capabilities of the SAX-based underlying processing framework, which is capable of piping several translators that are working concurrently. This concurrent processing facilitates the modularization of complex processing tasks in more affordable, simpler translators, which can be developed and maintained using separated translation schemas

Referencia de citas bibliográficas:

(Fischer et al., 1988), (ISO, 1997) , (Kawaguchi, 2002) , (Kodaganallur, 2004), (Paakki, 1995) , (Sierra et al., 2008a), (Sierra et al, 2008b) , (Stanchfield, 2005), (Trancón et al., 2003)

La dirección del artículo original es la siguiente:

http://doi.ieeecomputersociety.org/10.1109/DEXA.2008.32

6.2 XML Language-Oriented Processing with XLOP

Cita completa:

Sarasa-Cabezuelo, A., Temprado-Battad, B., Sierra-Rodríguez, J.L, Fernández-Valmayor, A. XML Language-Oriented Processing with XLOP. *5th International Symposium on Web and Mobile Information Services* (WAMIS'09) - 23rd *International Conference on Advanced Information Networking and Applications* (AINA'09), 26-29 de Mayo de 2009, Bradford, Inglaterra. AINA'09 Workshops. IEEE Computer Society, pp. 322-327

Resumen original de la contribución:

This paper presents XLOP (XML language-oriented processing), an environment that permits the description of XML processing applications with attribute grammars (formalism used to describe the semantics of computer languages). The environment also includes a generator able to translate attribute grammar-based specifications into the specification language of CUP, a well-known YACC-like environment for writing language processors. The final goal of XLOP is to facilitate the development and maintenance of XML processing applications, whilst preserving the flexibility of general-purpose XML processing models. Also, since it is based on attribute grammars, XLOP provides a higher abstraction level than other similar translation scheme-based approaches.

Referencia de citas bibliográficas:

(Aho et al., 2006) , (Appel ,1997), (Birceck et al., 2001), (Bray et al., 2008), (Gamma et al., 1994), (Kawaguchi, 2002), (Knuth, 1968), (Paakki, 1995) , (Sarasa et al., 2008a) , (Sierra et al., 2008b), (SAX, 2004), (Stanchfield, 2005), (DOM, 2000), (XMLPULL, 2004)

La dirección del artículo original es la siguiente:

http://doi.ieeecomputersociety.org/10.1109/WAINA.2009.17

6.3 Processing Learning Objects with Attribute Grammars

Cita completa:

Sarasa-Cabezuelo, A. , Sierra-Rodriguez, J.L, Fernández-Valmayor, A. Processing Learning Objects with Attribute Grammars. *Ninth IEEE International Conference on Advanced Learning Technologies (ICALT 2009)*. 15-17 de Julio de 2009, Riga, Letonia. IEEE Computer Society, pp. 527-531.

Resumen original de la contribución:

The services provided by learning object repositories are usually enabled by the processing of the metadata documents associated with these learning objects. This paper proposes a way to process these metadata documents, which are usually encoded in XML, through a framework called XLOP. XLOP is based on attribute grammars, a well-known technique used in the development of language processors. XLOP makes the automatic generation of efficient XML processing components from high-level specifications possible, enhancing the maintainability of the aforementioned services. The technique is illustrated in the context of Chasqui, a system for building repositories of learning objects in specialized domains.

Referencia de citas bibliográficas:

(Aho et al., 2006) , (Birceck et al., 2001), (Hatala et al., 2004), (LOM, 2002), (Knuth, 1968), (Lam et al., 2008), (Paakki, 1995) , (Sarasa et al., 2009a), (Sierra et al., 2006a) , (Sierra et al., 2007a), (Sierra et al., 2008a), (Sierra et al., 2008c)

La dirección del artículo original es la siguiente:

http://doi.ieeecomputersociety.org/10.1109/ICALT.2009.211

6.4 Procesamiento de Documentos XML Dirigido por Lenguajes en Entornos de E-Learning

Cita completa:

Sarasa-Cabezuelo, A, Sierra-Rodríguez, J.L, Fernández-Valmayor, A. Procesamiento de Documentos XML Dirigido por Lenguajes en Entornos de E-Learning. *IEEE RITA*, Agosto de 2009. Volumen 4. Numero 3. pp: 175-183.ISSN 1932-8540

Resumen original de la contribución:

This paper proposes the use of attribute grammars in order to systematize the processing of XML documents in e-Learning environments. For this purpose, it presents XLOP (XML Language-Oriented Processing), an XML processing environment based on this technique. It also illustrates how XLOP is used in two different e-Learning contexts: language-driven production of educational applications, and processing of metadata documents for reusable learning objects.

Referencia de citas bibliográficas:

(Aho et al., 2006) , (Appel ,1997), (Birceck et al., 2001), (Bork, 1985) , (Fernández-Manjón et al., 2008), (IMS, 2008), (Kawaguchi, 2002), (Knuth, 1968), (Paakki, 1995), (Sarasa et al., 2008a), (Sarasa et al., 2008b), (Sarasa et al., 2009a), (Sierra et al., 2006a), (Sierra et al., 2007b), (Sierra et al., 2007c), (Sierra et al., 2008a), (Sierra et al., 2008c), (Sierra et al., 2008d), (Stanchfield, 2005), (XTutor, 2007)

La dirección del artículo original es la siguiente:

http://rita.det.uvigo.es/200908/uploads/IEEE-RITA.2009.V4.N3.A2.pdf

6.5 A Generative Approach to the Construction of Application-Specific XML Processing Components

Cita completa:

Sarasa-Cabezuelo, A., Martínez-Avilés, A., Sierra-Rodríguez, J.L., Fernández-Valmayor, A. A Generative Approach to the Construction of Application-Specific XML Processing Components. *35th Euromicro Software Engineering and Advanced Applications Conference (SEAA'09)*. 27-29 de Agosto de 2009, Patras, Grecia.IEEE Computer Society, pp. 345-352

Resumen original de la contribución:

This paper proposes a generative approach to the construction of XML processing components. This approach promotes the high-level description of XML processing tasks with attribute grammars (a high-level formalism used in the definition of computer languages). The components themselves are produced by automatically processing these high-level specifications with a suitable generator. The approach substantially enhances the construction and maintenance of task-specific XML processing components compared to hand-coding or more rigid generative solutions. In order to illustrate the approach, we will show how XLOP (XML Language-Oriented Processing), an XML processing environment based on these concepts, is used for the development of an XML-based courseware system in the e-Learning domain.

Referencia de citas bibliográficas:

(Aho et al., 2006) , (Appel ,1997), (Birceck et al., 2001), (Coombs et al., 1987), (Czarnecki, 2000), (Gançarski et al., 2006), (Kawaguchi, 2002), (Kay, 2007), (Knuth, 1968), (Koch et al., 2007), (Lam et al., 2008), (Neven, 1999), (Neven, 2005), (Paakki, 1995), (Psaila et al., 1999), (Purdom, 1980), (Sarasa et al., 2008a), (Sarasa et al., 2009a), (Sierra et al., 2008a), (Stanchfield, 2005)

La dirección del artículo original es la siguiente:

http://doi.ieeecomputersociety.org/10.1109/SEAA.2009.14

6.6 Building an Enhanced Syntax-Directed Processing Environment for XML Documents by Combining StAX and CUP

Cita completa:

Sarasa-Cabezuelo, A., Temprado-Battad, B., Martínez-Avilés, A., Sierra-Rodriguez, J.L., Fernández-Valmayor, A. Building an Enhanced Syntax-Directed Processing Environment for XML Documents by Combining StAX and CUP. *Fourth International Workshop on Flexible Database and Information Systems Technology* (FlexDBIST-09) - *19th International Conference on Database and Expert Systems Application* (DEXA'09). 31 de Agosto a 4 de Septiembre de 2009, Linz, Austria. DEXA'09 Workshops. IEEE Computer Society, pp.427-431

Resumen original de la contribución:

Building on our previous work on the combination of JavaCC and SAX, we developed a new syntax-directed processing environment for XML documents. In this new environment, we use CUP as the parser-generation tool and StAX as the interface with the underlying XML parsing framework. The new environment supports a richer set of processing-specific context-free grammars (i.e., the grammars used for representing the processing-oriented structure of the processed documents) and enables the construction of more efficient and complex processors. In particular, the environment makes possible an event-driven computation style that enables processes that require multiple passes on the document tree, although such a tree is never explicitly built. This paper describes the environment and illustrates its use in the development of, an XML-driven courseware system.

Referencia de citas bibliográficas:

(Aho et al., 2006) , (Appel ,1997), (Kawaguchi, 2002), (Kodaganallur, 2004), (Lam et al., 2008), (Martínez-Ortiz, 2009), (Sarasa et al., 2008a), (Sierra et al., 2006a), (Sierra et al., 2008a), (Stanchfield, 2005)

La dirección del artículo original es la siguiente:

http://doi.ieeecomputersociety.org/10.1109/DEXA.2009.60

6.7 Engineering web services with attribute grammars: a case study

Cita completa:

Sarasa-Cabezuelo, A., Temprado-Battad, B., Sierra-Rodríguez, J.L. Engineering web services with attribute grammars: a case study. *ACM SIGSOFT Software Engineering Notes*, 24 de Enero de 2011. Volumen 36. Número 1. pp :1-8. DOI: 10.1145/1921532.1921545.

Resumen original de la contribución:

Applications based on web services make an extensive use of XML documents. These XML documents are structured according to different markup vocabularies, which represent information such as the inputs and the outputs of the services. This paper proposes a way of implementing web services using a framework called XLOP (XML Language-Oriented Processing). XLOP includes a declarative domain-specific language based on attribute grammars, a well-known declarative specification technique used in the development of language processors. XLOP makes possible the automatic generation of efficient XML-processing components from high-level, declarative specifications, facilitating the development of the aforementioned services and enhancing their maintainability. This technique is illustrated in the context of Chasqui, a system for building repositories of learning objects in specialized domains, which implements a REST web service for checking constraints on the metadata of the stored learning objects.

Referencia de citas bibliográficas:

(Aho et al., 2006) , (Birceck et al., 2001), (Bray et al., 2008), (Cerami, 2002), (Fielding, 2000), (Gançarski et al., 2006), (Knuth, 1968), (Knuth, 1971),(Lam et al., 2008), (Laurent et al., 2001), (Navarro et al., 2005), (Neven, 2005), (Okajima, 2002), (Paakki, 1995) , (Polsani , 2003), (Psaila et al., 1999), (Richardson et al., 2007),(Sarasa et al., 2008a), (Sarasa et al., 2009a), (Sarasa et al., 2009b), (Sierra et al., 2004) , (Sierra et al., 2006a), (Sierra et al., 2006b) ,(Sierra et al., 2008a), (Sierra et al., 2008c),(Stanchfield, 2005)

La dirección del artículo original es la siguiente:

http://dx.doi.org/10.1145/1921532.1921545

6.8 Building XML-Driven Application Generators with Compiler Construction Tools

Cita completa:

Sarasa-Cabezuelo, A., Temprado-Battad, B., Rodriguez-Cerezo, D., Sierra-Rodríguez, J.L. Building XML-Driven Application Generators with Compiler Construction Tools. *Computer Science and Information Systems.* Junio de 2012. Volumen 9, Número 2. pp: 485-504. ISSN: 1820-0214 (Índice de impacto JCR en 2011: 0.625)

Resumen original de la contribución:

This paper describes how to use conventional compiler construction tools, and parser generators in particular, to build XML-driven application generators. In our approach, the document interface is provided by a standard stream-oriented XML processing framework (e.g., SAX or StAX). This framework is used to program a generic, customizable XML scanner that transforms documents into streams of suitable tokens (opening and closing tags, character data, etc.). The next step is to characterize the syntactic structure of these streams in terms of generation-specific context-free grammars. By adding suitable semantic attributes and semantic actions to these grammars, developers obtain generation-oriented translation schemes: high-level specifications of the generation tasks. These specifications are then turned into working application generators by using standard parser generation technology. We illustrate the approach with <e-Subway>, an XML-driven generator of shortest-route search applications in subway networks.

Referencia de citas bibliográficas:

(Aho et al., 2006), (Appel, 1997), (Brownell, 2002), (Cleaveland, 1988), (Cleaveland, 2001), (Czarnecki, 2000), (Feng et al., 1993), (Fowler , 2010), (Fuentes-Fernández et al., 2009), (Gançarski et al., 2002), (Gançarski et al., 2006),(IMS, 2008), (Knuth, 1968), (Koch et al., 2007), (Kodaganallur, 2004) , (Lam et al., 2008), (Leite-Ramalho, 2000), (Marini, 2002), (McLaughlin, 2002), (McLaughlin, 2006), (Moreno-Ger et al., 2007), (Nakano, 2004), (Neven, 1999), (Neven, 2005), (Nishimura et al., 2005), (Okajima, 2002), (Paakki, 1995) , (Parr, 2007), (Parr, 2010), (Psaila et al., 1999), (Rodriguez-Cerezo et al., 2011b),(Sarasa et al., 2008a), (Sarasa et al., 2009a), (Sarasa et al., 2009d), (Sarasa et al., 2009e), (Schwentick, 2007), (Sierra et al., 2004), (Sierra et al., 2006b),(Sierra et al., 2008a), (Stanchfield, 2005), (Tidwell, 2008), (Vlist, 2003a), (Wallmsley, 2007).

La dirección del artículo original es la siguiente:

http://www.doiserbia.nb.rs/Article.aspx?id=1820-021412000025

6.9 The Grammatical Approach: A Syntax-Directed Declarative Specification Method for XML Processing Tasks

Cita completa:

Sarasa-Cabezuelo, A., Sierra-Rodríguez, J.L. The Grammatical Approach: A Syntax-Directed Declarative Specification Method for XML Processing Tasks. *Computer Standards & Interfaces. En prensa* (versión on-line 9 de Julio 2012). DOI: 10.1016/j.csi.2012.06.006. ISSN: 0920-5489. (Índice de impacto JCR en 2011: 1.255)

Resumen original de la contribución:

This paper describes the *grammatical* approach, an approach to the specification of XML processing tasks based on *attribute grammars*. This approach describes how to provide task-specific context-free grammars for XML documents, as well as how to decompose complex processing tasks into simpler ones with *attribute-grammar fragments*. The result is a high-level, syntax-directed declarative specification for the processing of XML documents, which facilitates the development and maintenance of complex XML processing applications while preserving the flexibility of general-purpose XML processing models. The grammatical approach is illustrated using *Chasqui*, an e-Learning platform for building educational digital libraries of *learning objects*.

Referencia de citas bibliográficas:

(Aho et al., 2006), (Akker et al., 1990), (Appel, 1997), (Birceck et al., 2001), (Bray et al., 2008), (Brownell, 2002), (Clark, 1999), (Clark et al., 1999), (DOM, 2000) , (Draper et al. 2003), (Dueck et al., 1990), (Farrow et al., 1992), (Feng et al., 1993), (Fülop, 1981), (Fülop et al., 1998), (Gançarski et al., 2006), (Ganzinger et al., 1984), (Goldfarb, 1991), (Grune et al., 2008), (Hedin, 1989), (Hedin, 1999), (Kastens et al., 1994), (Knuth, 1968), (Knuth, 1971), (Kennedy et al., 1979), (Koch et al., 2007), (Kodaganallur, 2004), (Lam et al., 2008), (McLaughlin, 2002), (Murata et al., 2005), (Nakano, 2004), (Neven, 1999), (Neven, 2005), (Nishimura et al., 2005), (Okajima, 2002), (Paakki, 1995), (Parr, 2007), (Polsani , 2003), (Psaila et al., 1999),(Rebernak et al., 2006) ,(Saraiva et al., 1999), (Sarasa et al., 2008a), (Sarasa et al., 2009a), (Sarasa et al., 2009b), (Sarasa et al., 2009d), (Sarasa et al., 2009e), (Sarasa et al., 2011), (Sarasa et al., 2012a), (Schwentick, 2007), (Sethi, 1997), (Sierra et al., 2006a), (Sierra et al., 2008c), (Stanchfield, 2005), (Temprado-Battad, et al., 2010), (Vlist, 2003a), (Vogt et al., 1989), (Wyk et al., 2002)

La dirección del artículo original es la siguiente:

http://dx.doi.org/10.1016/j.csi.2012.06.006

Bibliografía

(Abiteboul et al., 2000) Abiteboul, S., Buneman, P., Suciu, D. (2000). Data on the Web: From Relations to Semistructured Data and XML. *Morgan Kaufmann*.

(Ablas, 1991) Ablas, H. (1991). Attribute Evaluation Methods. *Attribute Grammars, Applications and Systems. Lecture Notes in Computer Science. Springer,* 545, pp. 48-113.

(Adler et al., 2001) Adler, S., Berglund, A., Caruso, J., Deach, S., Graham, T., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., Zilles, S. (2001). Extensible Stylesheet Language (XSL) Version 1.0. *W3C Recommendation*.

(Akker et al., 1990) Akker, R., Melichar, B., Tarhio, J. (1990). The Hierarchy of LR-attributed grammars. International Workshop on Attribute Grammars and their Applications (WAGA'90). *Attribute Grammars and their Applications.Lecture Notes in Computer Science. Springer*, 461, pp. 13-28.

(Akker et al., 1991) Akker, R., Melichar, B., Tarhio, J. (1991). Attribute Evaluation and Parsing. *Attribute Grammars, Applications and Systems. Lecture Notes in Computer Science. Springer*, 545, pp. 187-214.

(Ahmed et al., 2001) Ahmed, K., Ancha, S., Cioroianu, A., Cousins, J., Crosbie, J., Davies, J., Gabhart, K., Gould, S., Laddad, R., Li, S., Macmillan, B., Rivers-Moore, D., Skubal, J., Watson, K., Willians, S., Hart, J. (2001). Professional Java XML. *Wrox Press*.

(Aho et al., 2006) Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D. (2006). Compilers: principles, techniques and tools (2nd Edition). *Addison-Wesley*.

(Apparao et al., 1998) Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Le Hors, A., Nicol, G., Robie, J., Sutor, R., Wilson, C., Wood, L. (1998).Document Object Model Level 1. *W3C Recommendation*.

(Appel, 1997) Appel, A.W. (1997). Modern Compiler Implementation in Java. *Cambridge University Press.*

(Augusteijn, 1990) Augusteijn, L. (1990). The Elegant Compiler Generator System. International Workshop on Attribute Grammars and their Applications (WAGA'90). *Attribute Grammars, Applications and Systems. Lecture Notes in Computer Science. Springer* , 461, pp. 238-254.

(Backus, 1959) Backus, J.W. (1959). The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. *International Conference on Information Processing*. UNESCO, pp. 125-132.

(Berners-Lee et al., 2005) Berners-Lee, T., Fielding, R., Masinter, L. (2005). Uniform Resource Identifier (URI): Generic Syntax [RFC3986]. *Internet Engineering Task Force (IETF).*

(Birceck et al., 2001) Birbeck, M., Duckett, J., Gudmundsson, O.G., Kobak, P., Lenz, E., Livingstone, S., Marcus, D., Mohr, S., Pinnock, J., Visco, K., Watt, A., Willians, K., Zaev, Z., Ozu, N. (2001). Professional XML 2nd Edition. *Wrox Press.*

(Biron et al., 2001) Biron, P.V., Malhotra, A. (2001). XML Schema Part 2: Datatypes. *W3C Recommendation*.

(Bischoff, 1992) Bischoff, K.M. (1992). Design, Implementation, Use and Evaluation of Ox: An Attribute-Grammar Compiling System based on Yacc, Lex and C. Technical Report #92-31, Dp. Of Computer Science. University of Iowa State.

(Bochmann, 1976) Bochmann, G. (1976). Semantic Attributes for Grammars with Regular Expressions. Technical Report #195. Université de Montréal.

(Bork, 1985) Bork, A. (1985).Personal Computers for Education. *Harper & Rows*.

(Bos et al., 1998) Bos, B., Lie, W.H., Lilley, C., Jacobs, I. (1998). Cascading Style Sheets, Level 2 CSS2 Specification. *W3C Recommendation*.

(Bourret, 2003) Bourret, R. (2003). XML Data Binding Resources. *www.rpbourret.com/xml/XMLDataBinding.htm*.

(Brand et al., 2001) Brand, M.G.J v.d. Deursen, A, v. Heering, J. Jong, H.A.d. Jonge, M.d. Kuipers, T. Klint, P. Moonen, L. Olivier, P.A. Scheerder, J. Vinju, JJ. Visser, E. Visser, J. (2001). The Asf +Sdf Meta-environment: A Component-Based Language Development Environment. *10th International Conference on Compiler Construction(CC'01)*. Lecture Notes in Computer Science. Springer, pp. 365-370.

(Bray et al., 2008) Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C Recommendation*.

(Bray et al., 2009) Bray, T., Hollander, D., Layman, A., Tobin, R., Thompson, H.S. (2009). Namespaces in XML 1.0(Third Edition). *W3C Recommendation*.

(Brosgol, 1974) Brosgol, M. B. (1974). Deterministic Translation Grammars. PhD. Thesis. University of Harvard.

(Brownell, 2002) Brownell, D. (2002). SAX2. *O'Relly*.

(Bork, 1985) Bork, A. (1985). Personal Computers for Education. Harper & Rows.

(Brüggemann-Klein et al., 1998) Brüggemann-Klein, A., Wood, D. (1998). One-Unambiguous Regular Languages. *Information and Computation*, 142(2), pp. 182-206.

(Cerami, 2002) Cerami, E. (2002). Web Services Essentials. *O'Relly Media.*

(Czarnecki, 2000) Czarnecki, K. (2000). Generative Programming: Methods, tools and Applications. *Addison-Wesley*.

(Chang, 2003) Chang, B. (2003). Document Object Model Level 3 Validation. *W3C Candidate Recommendation*.

(Clark, 1999) Clark, J. (1999). XSL Transformations (XSLT) Version 1.0. *W3C Recommendation*.

(Clark et al., 1999) Clark, J., DeRose, S. (1999). XML Path Language (XPath) Version 1.0. *W3C Recommendation*.

(Clark, 2001a) Clark, J. (2001). TREX – Tree Regular Expressions for XML Language Specification. *Thai Open Source Software Center*.

(Clark, 2001b) Clark, J. (2001). TREX – Tree Regular Expressions for XML Tutorial. *Thai Open Source Software Center*.

(Clark et al., 2001a) Clark, J., Makoto, M. (2001). Relax NG Specification. *Oasis Committee Specification*.

(Clark et al., 2001b) Clark, J., Makoto, M. (2001). Relax NG Tutorial. *Oasis Committee Specification*.

(Cleaveland, 1988) Cleaveland, J.C. (1998). Building Application Generators. *IEEE Software*, 5(4), pp. 25-33.

(Cleaveland, 2001)Cleaveland, J.C. (2001). Program Generators with XML and Java. *Prentice Hall*.

(Cohen et al., 1979) Cohen, R., Harry, E. (1979). Automatic Generation of near-optimal linear-time Translators for Non-circular Arttribute Grammars. *6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages(POPL'79)*. ACM, pp. 121-134

(Comon et al., 1997) Comon, H., Dauchet, M., Gilleron, R., Jacquermard, F., Lugiez, D., Tison, S., Tommasi, M. (1997). Tree Automata Techniques and Applications. *Technical Report. Groupe de Recherche sur l'Apprentissage Automatique. Univ. Lille*.

(Coombs et al., 1987) Coombs, J. H., Renear, A. H., DeRose, S. J. (1987). Markup Systems and the Future of Scholarly Text Processing. *Communications of the ACM*, 30 (11), pp. 933-947.

(Cowan et al., 2001) Cowan, J., Tobin, R. (2001). XML Information Set. *W3C Recommendation*.

(Dabek et al., 2002) Dabek, F., Zeldovich, N., Kaashoek, M. F., Mazières, D., Morris, R. (2002). Event-driven Programming for Robust Software. *10th ACM SIGOPS European Workshop*. ACM, pp. 22-25.

(DOM, 2000) Document Object Model Technical Reports. *W3C Recommendations*.

(Draper et al. 2003) Draper, D., Fankhauser, P., Fernández, M., Malhotra, A., Rose, K., Rys, M., Siméon, J., Wadler, P. (2003). XQuery 1.0 and XPath 2.0 Formal Semantics. *W3C Working Draft*.

(Dueck et al., 1990) Dueck, G. D. P., Cormack, G. V. (1990). Modular Attribute Grammars. *Computer Journal,* 33(2), pp. 164-172.

(Ekman et al., 2007) Ekman, T., Hedin, G. (2007). The JastAdd system - modular extensible compiler construction. *Science of Computer Programming*, 69 (1-3), pp. 14-26.

(Fallside, 2001) Fallside, D.C. (2001). XML Schema Part 0: Primer. *W3C Recommendation*.

(Farrow et al., 1992) Farrow, R., Marlowe, T. J., Yellin, D. M. (1992). Composable Attribute Grammars: Support for Modularity in Translator Design and Implementation. *19th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'92)*. ACM, pp. 223-234.

(Feng et al., 1993) Feng, A., Wakayama, T. A. (1993).Grammar-based Transformation System for Structured Documents. *Electronic Publishing*, 6(4), pp. 361-372.

(Fernández-Manjón et al., 2008) Fernández-Manjón, B., Sierra, J.L., Moreno-Ger, P., Martínez-Ortiz, I. (2008). Uso de Estándares Aplicados a TIC en Educación. Informe Técnico 16. *Centro Nacional de Información y Comunicación Educativa (CNICE).*

(Fielding, 2000) Fielding, R.T. (2000). Architectural Styles and the Design of Network-based Software Architectures, PhD thesis. *University of California*.

(Fischer et al., 1988) Fischer, C.N., LeBlanc, R.J. (1998). Crafting a Compiler. *Benjaming/Cummings.*

(Fowler, 2010) Fowler, M. (2010). Domain Specific Languages. *Addison-Wesley*.

(Fuchs, 1997) Fuchs, M. (1997). Domain Specific Languages for ad hoc Distributed Applications. *Conference on Domain-Specific Languages on Conference on Domain-Specific Languages(DSL'97), pp. 27-36.*

(Fülop, 1981) Fülop, Z. (1981). On Attributed Tree Transducers, *Acta Cybernetica*, 5(3), pp. 261-279.

(Fülop et al., 1998) Fülop, Z., Vogler, H. (1998). Syntax-Directed Semantics: Formal Models Based on Tree Transducers. *Theoretical Computer Science. EATCS Series.*

(Fuentes-Fernández et al., 2009) Fuentes-Fernández, R., Gómez-Sanz, J., Pavón J. (2009). Requirements Elicitation and Analysis of Multiagent Systems Using Activity Theory. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(2), pp. 282-298.

(Gamma et al., 1994) Gamma, E., Helm, R., Jhonson, R., Vlissides, J. M. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley.*

(Gançarski et al., 2002) Gançarski, A. L., Doucet, A., Henriques, P. R. (2002). Information Retrieval from Structured Documents Represented by Attribute Grammars .*International Conference on Information Systems Modelling*.

(Gançarski et al., 2006) Gançarski, A. L., Doucet, A., Henriques, P. R. (2006). Attribute grammar-based Interactive System to Retrieve Information from XML Documents. *IEEE Proceedings-Software*, 153(2), pp. 51-60.

(Ganzinger et al., 1984) Ganzinger, H., Giegerich, R. (1984). Attribute Coupled Grammars. *1984 SIGPLAN symposium on Compiler construction(SIGPLAN'84). ACM SIGPLAN Notices, 19(6),* pp. 157 - 170.

(Goldfarb, 1991) Goldfarb, C. (1991). The SGML Handbook. *Oxford University Press*.

(Gracia-Benitez, 2010) Gracia, J.P. (2010). Marco para la Transformación de Modelos basado en Gramáticas de Atributos. Proyecto Fin de Máster en Sistemas Inteligentes. Facultad de Informática, UCM.

(Gray et al., 1992) Gray, R.W., Heuring, V.P., Levi, S.P., Sloane, A.M., Waite, W.M. (1992). Eli: A Complete, Flexible Compiler Construction System. *Communications of the ACM*, 35(2), pp. 121-131.

(Grune et al., 2008) .Grune, D., Jacobs, C.J.H. (2008). Parsing Tecniques – A Practical Guide 2nd Edition. *Monographs in Computer Science. Springer*.

(Hatala et al., 2004) Hatala M., Richards G., Eap T., Willms J. (2004). The Interoperability of Learning Object Repositories and Services: Standards, Implementations and Lessons Learned. *13th international World Wide Web Conference(www2004)*. ACM, pp. 19-27.

(Havasi, 2002) Havasi, F. (2002). XML Semantics Extension. *Acta Cybernetica* , 15(2), pp. 509-528.

(Hedin, 1989) Hedin, G. (1989). An Object-Oriented Notation for Attribute Grammars. *Third European Conference on Object-Oriented Programming* (ECOOP'89). Cambridge University Press ,pp. 329-345.

(Hedin, 1999) Hedin, G. (1999). Reference Attribute Grammars. *2nd International Workshop on Attribute Grammars and their Applications (WAGA'99)*, pp. 153-172.

(Hégaret, 2003) Le Hégaret, P. (2003). Document Object Model Level 3 Events. *W3C Working Draft*.

(Henriques et al., 2005) Henriques, P.R., Varanda-Pereira, M.J., Mernik, M., Lenic, M., Gray, J.G., Wu, H. (2005). Automatic Generation of Language-Based Tools using the LISA System. *IEEE Proceedings – Software*, 152(2), pp. 54-69.

(Hors et al., 2000a) Le Hors, A., Le Hégaret, P., Wood, L., Nicol, G., Robie, J., Champion, M., Byrne, S. (2000). Document Object Model Level 2 Core. *W3C Recommendation.*

(Hors et al., 2000b) Le Hors, A., Cable, L. (2000). Document Object Model Level 2 Views. *W3C Recommendation.*

(Hors et al., 2003) Le Hors, A., Le Hégaret, P. (2003). Document Object Model Level 3 Core (Eds). *W3C Working Draft.*

(Hudson, 1999) Hudson, S.E. (1999). CUP User's Manual. *http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html*.

(IMS, 2008) IMS Question and Test Interoperability 2.1. *www.imsglobal.org/question/*

(ISO, 1996) *ISO/IEC Standard 14977*. *Information technology -- Syntactic metalanguage -- Extended BNF*.

(ISO, 1997) *ISO/IEC Standard 10744. Hypermedia/Time-based Structuring Language (HyTime)*, 2nd Ed.

(JavaBeans, 1997) JavaBeansTM version 1.01. *Sun Microsystems.*

(Jalili,1983) Jalili, F. (1983). A General Linear-Time Evaluator for Attribute Grammars. *ACM SIGPLAN Notices,* 18(9), pp. 35-44.

(Jelliffe, 2001) Jelliffe, R. (2001). The W3C XML Schema Specification in Context. *O'Reilly XML.com.*

(Jelliffe, 2002) Jelliffe, R. (2002). The Schematron Assertion Language 1.5. *Academia Sinica Computing Centre*.

(Jones, 1990) Jones, L.G. (1990). Efficient Evaluation of Circular Attribute Grammars. *ACM Transactions of Programming Languages and Systems*, 12(3), pp. 429-462.

(Jourdan, 1984) Jourdan, M. (1984). Strongly non-circular Attribute Grammars and their Recursive Evaluation. *1984 SIGPLAN symposium on Compiler construction(SIGPLAN'84).* ACM SIGPLAN Notices, 19(6), pp. 81-93.

(Jourdan et al., 1991) Jourdan, M., Parigot, D. (1991). Internals and Externals of the FNC-2 Attribute Grammar System. *Attribute Grammars, Applications and Systems. Lecture Notes in Computer Science. Springer*, 545, pp. 485-504.

(Jullig et al., 1984) Jullig, R.K., DeRemer, F. (1984). Regular Right-Part Attribute Grammars. *1984 SIGPLAN Simposium on Compiler Construction (SIGPLAN'84).* ACM SIGPLAN Notices, 19(6), pp. 171-178.

(Kastens, 1980) Kastens, U. (1980). Ordered Attribute Grammars. *Acta Informatica*, 13, pp. 229-256.

(Kastens et al, 1982) Kastens, U., Hutt, B., Zimmermann, E. (1982). GAG: A Practical Compiler Generator. Lecture Notes in Computer Science. Springer, 141.

(Kastens et al., 1994) Kastens, U., Waite, W. M. (1994). Modularity and Reusability in Attribute Grammars. *Acta Informatica*, 31(7), pp. 601-627.

(Kay, 2007) Kay, M. (2007). XSL Transformations (XSLT) Version 2.0. *W3C Recommendation*.

(Kawaguchi, 2002) Kawaguchi, K. (2002). Flexible Data-Biding with RelaxNGCC. *Extreme Markup Languages 2002*.

(Kennedy et al., 1979) Kennedy, K., Ramanathan, J. (1979). A Deterministic Attribute Grammar Evaluator Based on Dynamic Sequencing. *ACM Transaction of Programming Languages and Systems*, 1(1), pp. 142-160.

(Kesselman et al., 2000) Kesselman, J., Robie, J., Champion, M., Sharpe, P., Apparao, V., Wood, L. (2000). Document Object Model Level 2 Traversal and Range. *W3C Recommendation.*

(Klint et al., 2005) Klint, P. Lämmel, R. Verhoef, C. (2005). Toward an Engineering Discipline for Grammarware. *ACM Transactions on Software Engineering and Methodology*, 14(3), pp. 331-380.

(Knuth, 1965) Knuth, D.E. (1965). On the Translation of Languages from Left to Right. *Information & Control*, 8 , pp. 607-639.

(Knuth, 1968) Knuth, D. E. (1968). Semantics of Context-free Languages. *Mathematical System Theory*, 2(2), pp. 127-145.

(Knuth, 1971) Knuth, D.E. (1971). Correction: Semantics of Context-Free Languages, *Mathematical Systems Theory*, 5(1), pp. 95-96.

(Koch et al., 2007) Koch, C. Scherzinger, S. (2007). Attribute Grammars for Scalable Query Processing on XML Streams. *The VLDB Journal*, 16(3), pp. 317-342.

(Kodaganallur, 2004) Kodaganallur, V. (2004). Incorporating Language Processing into Java Applications: A JavaCC Tutorial. *IEEE Software*, 21(4), pp. 70-77.

(LaLonde, 1976) LaLonde, W.R. (1976). Regular Right-Part Grammars and their Parsers. *Communications of the ACM*, 20(10), pp. 731-741.

(Lam et al., 2008) Lam, T.C., Ding, J.J., Liu, J.C. (2008). XML Document Parsing: Operational and Performance Characteristics. *IEEE Computer*, 41(9), pp. 30-37.

(Laurent et al., 1999) Laurent, S.St., Cerami, S.E. (1999). Building XML applications. *McGraw-Hill.*

(Laurent et al., 2001) Laurent, S., Jhonston, J., Dumbill, E. (2001). Programming Web Services with XML-RPC, *O'Relly Media.*

(Lee et al., 2000) Lee, D., Chu, W.W. (2000). Comparative Analysis of Six XML Schema Languages. *ACM SIGMOD Record*, 29(3), pp. 76-87.

(Leite-Ramalho, 2000) Leite-Ramalho, J.C. (2000) Anotação Estrutural de Documentos e sua Semântica -- Especificação da Sintaxe, Semântica e Estilo para Documentos. *PhD Thesis*. Departamento de Informática, Universidade do Minho.

(Levine, 2009). Levine, J. (2009). Flex & Bison: Text Processing Tools. *O'Reilly Media.*

(LOM, 2002) IEEE Standard 1484.12.1-2002. *IEEE Standard for Learning Object Metadata.*

(Magnusson et al. 2007) Magnusson, E., Hedin, G. (2007). Circular Reference Attributed Grammars—Their Evaluation and Applications. *Science of Computer Programming*, 68(1), 21-37.

(Marini, 2002) Marini, J. (2002). Document Object Model: Processing Structured Documents. *McGraw-Hill.*

(Martínez-Ortiz, 2009) Martínez-Ortiz, I., Sierra, J.L., Fernández-Manjón, B. (2009). Authoring and Reengineering of IMS Learning Design Units of Learning. *IEEE Transactions on Learning Technologies*, 2(3), pp. 189-202.

(McLaughlin, 2002) McLaughlin, B. (2002). Java & XML Data Binding. *O'Reilly Media.*

(McLaughlin, 2006) McLaughlin, B. (2006). Java & XML. *O'Reilly Media.*

(Megginson et al, 1998) Megginson, D., Leventhal, M., Ducharme, R. (1998). Structuring XML Documents. *Prentice-Hall*.

(Mordani et al., 2002) Mordani, R., Boag, S. (2002). Java API for XML Processing Version 1.2. *Sun Microsystems.*

(Moreno-Ger et al., 2007) Moreno-Ger, P., Sierra, J.L., Martínez-Ortiz, I., Fernández-Manjón, B. (2007). A Documental Approach to Adventure Game Development. *Science of Computer Programming*, 67(1), pp. 3-31.

(Murata, 1995) Murata, M. (1995). Forest Regular Languages and Tree Regular Languages. Technical Report. *Fuji Xerox Systems*.

(Murata, 1999) Murata, M. (1999). Hedge Automata: A Formal Model for XML Schemata. Technical Report. *Fuji Xerox Systems*.

(Murata, 2000) Murata, M. (2000). Regular Language Description for XML (RELAX). *ISO/IEC Technical Report* DTR-22250-1.2000.

(Murata et al., 2001) Murata, M., Lee, D., Mani, M. (2001). Taxonomy of XML Schema Languages Using Formal Language Theory.*Extreme Markup Languages 2001*.

(Murata et al., 2005) Murata, M., Lee, D., Mani, M., Kawaguchi, K. (2005). Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4), pp. 660-704.

(Nakano, 2004) Nakano, K. (2004). An Implementation Scheme for XML Transformation Languages Through Derivation of Stream Processors. *Programming Languages and Systems: Second Asian Symposium (APLAS'04).* Lecture Notes in Computer Science. Springer, 3302, pp. 74-90.

(Navarro et al., 2005) Navarro, A., Sierra, J.L., Fernández-Valmayor, A., Hernanz, H. (2005). From Chasqui to Chasqui II: an Evolution in the Conceptualization of Virtual Objects. *Journal of Universal Computer Science,* 11(9), pp. 1518-1529.

(Neven, 1999) Neven, F. (1999). Extensions of Attribute Grammars for Structured Document Queries, in *7th International Workshop on Database Programming Languages (DBLP'99)*. Lecture Notes In Computer Science. Springer, pp. 99-116.

(Neven, 2005) Neven, F. (2005). Attribute Grammars for Unranked Trees as a Query Language for Structured Documents*, Journal of Computer and System Sciences*, 70(2), pp. 221-257.

(Nishimura et al., 2005) Nishimura, S., Nakano, K. (2005). XML Stream Transformer Generation through Program Composition and Dependency Analysis. *Science of Computer Programming*, 54(2-3), pp. 257-290.

(Nymeyer, 1995) Nymeyer, A. (1995). A Grammatical Specification of Human-Computer Dialogue. *Computer Languages,* 21(1), pp. 1-16.

(Okajima, 2002) Okajima, D. (2002). RelaxNGCC -- *Bridging the Gap Between Schemas and Programs*, vol. 8 May 2002. *O'Reilly XML.com.*

(Paakki, 1991) Paakki, J. (1991). PROFIT: A System Integrating Logic and Attribute Grammars. *3rd International Symposium on Programming Language Implementation and Logic Programming (PLILP'91).* Lecture Notes in Computer Science. Springer, 528, pp. 243-254.

(Paakki, 1995) Paakki, J. (1995). Attribute Grammar Paradigms – A High-Level Methodology in Language Implementation. *ACM Computer Surveys*, 27(2), pp. 196-255.

(Parr, 2007) Parr, T. (2007). The Definitive ANTLR Reference: Building Domain-Specific Languages. *Pragmatic Bookshelf*.

(Parr, 2010) Parr, T. (2010). Language Implementation Patterns. *Pragmatic Bookshelf*.

(Parr et al. 2011) Parr, T., Fisher, K. (2011). LL(*): The Foundation of the ANTLR Parser Generator. *32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'11*). ACM SIGPLAN Notices*,* 46(6), pp. 425-436.

(Pixle, 2000) Pixle, T. (2000). Document Object Model Level 2 Events. *W3C Recommendation.*

(Polsani, 2003) Polsani, P. (2003). Use and Abuse of Reusable Learning Objects, *Journal of Digital Information*, 3(4).

(Prud'hommeaux, 2001) Prud'hommeaux, E. *XML* Processing Pipeline Model. XML processing Model Workshop. W3C. June 12-13. 2001

(Psaila et al., 1999) Psaila, G., Crespi-Reghizzi, S. (1999). Adding Semantics to XML. *2nd International Workshop on Attribute Grammars and their Applications (WAGA'99)*, pp. 113-132.

(Purdom, 1980) Purdom, P., Brown, C.A. (1980). Semantic Routines and LR(k) parsers. *Acta Informatica ,*14, pp. 299-315.

(Raggett, 1999) Raggett, D. (1999). Assertion Grammars. http://www.w3.org/People/Raggett/dtdgen/Docs/

(Rebernak et al., 2006) Rebernak, D., Mernik, M., Henriques, P. R. (2006). AspectLISA: an Aspect-oriented Compiler Construction Systems based on Attribute Grammars. *6th Workshop on Language Description Tools and Applications (LDTA'06).* Electronic Notes in Theoretical Computer Science. Elsevier, 164(2), pp. 37-56.

(Richardson et al., 2007) Richardson, L., Ruby, S. (2007).RESTFull Web Services. *O'Relly Media.*

(Rodríguez-Cerezo et al. 2011a) Rodriguez-Cerezo. D., Gómez-Albarrán, M. Sierra, J.L. (2011). From Collections of Exercises to Educational Games: A Process Model and a Case Study. *11th International Conference on Advanced Learning Technologies(ICALT 2011)*. IEEE Computer Society, pp. 282-284.

(Rodriguez-Cerezo et al., 2011b) Rodriguez-Cerezo, D., Sarasa-Cabezuelo, A., Sierra-Rodriguez, J.L. (2011). Implementing attribute grammars using conventional compiler construction tools. *Federated Conference on Computer Science and Information Systems (FedCSIS 2011)*. IEEE Computer Society, pp. 855-862.

(Rodriguez-Cerezo et al., 2012a) Rodriguez-Cerezo, D., Sarasa-Cabezuelo, A. Gómez-Albarrán, M. Sierra-Rodriguez, J.L. (2012). Facilitating Comprehension of Basic Concepts in Computer Language Implementation Courses: A Game-Based Approach. XIV Simposio Internacional en Informática Educativa (SIIE'12), 29-31 Octubre, 2012, Andorra.

(Rodriguez-Cerezo et al., 2012b) Rodriguez-Cerezo, D., Sarasa-Cabezuelo, A., Sierra-Rodriguez, J.L. (2012). A Systematic Approach to the Implementation of Attribute Grammars with Conventional Compiler Construction Tools. *Computer Science and Information Systems*. *En prensa*. DOI: 10.2298/CSIS111223022R.

(Saraiva et al., 1999) Saraiva, J., Swierstra D. (1999). Generic Attribute Grammars. *2nd International Workshop on Attribute Grammars and their Applications (WAGA'99)*, pp. 185-204.

(Sarasa et al., 2008a) Sarasa, A., Navarro, I., Sierra, J.L., Fernández-Valmayor, A. (2008). Building a Syntax Directed Processing Environment for XML Documents by Combining SAX and JavaCC. *18th International Workshop on Database and Expert Systems Application (DEXA'08)*. IEEE Computer Society, pp. 256-260.

(Sarasa et al., 2008b) Sarasa, A., Sierra, J.L., Fernández-Valmayor, A. (2008). Procesamiento de documentos XML dirigido por lenguajes en entornos de e-Learning. X *Simposio Internacional de Informática Educativa (SIIE'08)*, pp. 62-70.

(Sarasa et al., 2009a) Sarasa-Cabezuelo, A., Temprado-Battad, B., Sierra-Rodríguez, J.L., Fernández-Valmayor, A. (2009). XML Language-Oriented Processing with XLOP. *International Conference on Advanced Information Networking and Applications Workshops. (WAINA'09)*. IEEE Computer Society, pp. 322-327.

(Sarasa et al., 2009b) Sarasa-Cabezuelo, A., Sierra-Rodriguez, J.L., Fernández-Valmayor, A. (2009). Processing Learning Objects with Attribute Grammars. *9th IEEE International Conference on Advanced Learning Technologies (ICALT 2009)*. IEEE Computer Society, pp. 527-531.

(Sarasa et al., 2009c) Sarasa-Cabezuelo, A., Sierra-Rodríguez, J.L., Fernández-Valmayor, A. (2009) Procesamiento de Documentos XML Dirigido por Lenguajes en Entornos de E-Learning. *IEEE RITA*, 4(3) pp. 175-183.

(Sarasa et al., 2009d) Sarasa-Cabezuelo, A., Martínez-Avilés, A., Sierra-Rodríguez, J.L., Fernández-Valmayor, A. (2009). A Generative Approach to the Construction of Application-Specific XML Processing Components. *35th Euromicro Software Engineering and Advanced Applications Conference (SEAA'09)*. IEEE Computer Society, pp. 345-352.

(Sarasa et al., 2009e) Sarasa-Cabezuelo, A., Temprado-Battad, B., Martínez-Avilés, A., Sierra-Rodriguez, J.L., Fernández-Valmayor, A. (2009). Building an Enhanced Syntax-Directed Processing Environment for XML Documents by Combining StAX and CUP. *19th International Workshop on Database and Expert Systems Application (DEXA'09)*. IEEE Computer Society, pp. 427-431.

(Sarasa et al., 2011) Sarasa-Cabezuelo, A., Temprado-Battad, B., Sierra-Rodríguez, J.L. (2011). Engineering web services with attribute grammars: a case study. *ACM SIGSOFT Software Engineering Notes*, 36(1), pp. 1-8.

(Sarasa et al., 2012a) Sarasa-Cabezuelo, A., Temprado-Battad, B., Rodriguez-Cerezo, D., Sierra-Rodríguez, J.L. (2012). Building XML-Driven Application Generators with Compiler Construction Tools. *Computer Science and Information Systems*, 9(2), pp. 485-504.

(Sarasa et al., 2012b) Sarasa-Cabezuelo, A., Sierra-Rodríguez, J.L. (2012). The Grammatical Approach: A Syntax-Directed Declarative Specification Method for XML Processing Tasks. *Computer Standards & Interfaces. En prensa*. DOI: 10.1016/j.csi.2012.06.006

(SAX, 2004) Simple API for XML project. *www.saxproject.org*

(Schwentick, 2007) T. Schwentick, T. (2007). Automata for XML - A Survey. *Journal of Computer and System Sciences*, 73(3), pp. 289-315.

(Scott et al., 2006) Scott, E., Johnstone, A. Right nulled GLR parsers. *ACM Transactions on Programming Languages and Systems*, 28(4), pp. 577-618.

(Sethi, 1997) Sethi, R. (1997). Programming Languages Concepts & Constructs, 2nd ed. *Pearson Education*.

(Sierra, 2004) Sierra, J.L. (2004). Hacia un Paradigma Documental de Desarrollo de Aplicaciones. Tesis Doctoral. Facultad de Informática. UCM.

(Sierra et al., 2004) Sierra, J.L., Fernández-Valmayor, A., Fernández-Manjón, B., Navarro, A. (2004). ADDS--A Document-Oriented Approach for Application Development. *Journal of Universal Computer Science*, 10(9), pp. 1302-1324.

(Sierra et al., 2005) Sierra, J.L., Navarro, A. , Fernández-Manjón, B. , Fernández-Valmayor, A. (2004). Incremental definition and operationalization of domain-specific markup languages in ADDS. *SIGPLAN Notices* , 40(12), pp. 28-37.

(Sierra et al., 2006a) Sierra, J.L., Fernández-Valmayor, A., Guinea, M., Hernánz, H. (2006). From Research Resources to Virtual Objects: Process model and Virtualization Experiences. *Journal of Ed. Tech. & Society*, 9(3), pp. 56-68.

(Sierra et al., 2006b) Sierra J.L., Fernández-Valmayor A., Fernández-Manjón B. (2006). A Document-Oriented Paradigm for the Construction of Content- Intensive Applications. *Computer Journal*, 49(5), pp. 562-584.

(Sierra et al., 2007a) Sierra, J.L, Fernández-Valmayor, A. (2007). Universalizing Chasqui Repositories with a Flexible Importation / Exportation System. *Computers and Education: E-learning - from theory to practice*. *Springer*, pp. 99-110.

(Sierra et al., 2007b) Sierra, J.L., Fernández-Manjón, B., Fernández-Valmayor, A. (2007). Language-Driven Development of Web-Based Learning Applications. *6th International conference on web-based learning (ICWL 2007)*. Springer, pp. 520-531.

(Sierra et al., 2007c) Sierra, J.L., Fernández-Valmayor, A., Fernández-Manjón, B. (2007). How to Prototype an Educational Modeling Language. IX *Simposio Internacional de Informática Educativa (SIIE'07)*, pp. 97-102.

(Sierra et al., 2008a) Sierra, J.L., Fernández-Valmayor, A., Fernández-Manjón, B. (2008). From Documents to Applications Using Markup Languages. *IEEE Software,* 25(2), pp. 68-76.

(Sierra et al., 2008b) Sierra, J.L., Fernández-Pampillón, A., Fernández-Valmayor, A. (2008). An Environment for Supporting Active Learning in Courses on Language Processing. *13th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE 2008). ACM SIGCSE Bulletin, 40(3),* pp. 128-132.

(Sierra et al., 2008c) Sierra, J.L., Fernández-Valmayor, A. (2008). Tagging Learning Objects with Evolving Metadata Schema. *8th IEEE International Conference on Advanced Learning Technologies(ICALT'08).* IEEE Computer Society, pp. 829-833.

(Sierra et al., 2008d) Sierra, J.L., Fernández-Manjón, B., Fernández-Valmayor, A. (2008). A Language-Driven Approach for the Design of Interactive Applications. *Interacting with Computers*, 20(1), pp. 112-127.

(Stanchfield, 2005) Stanchfield, S. (2005). ANTXR: Easy XML Parsing based on The ANTLR Parser Generator. *JavaDude.com.*

(Stahl et al., 2006) Stahl, T., Voelter, M., Czarnecki, K. (2006). Model-Driven Software Development: Technology, Engineering, Management. *Wiley*.

(Schreiner et al., 1985) Schreiner, A.T., Friedman, H.G. (1985). Introduction to Compiler Construction with UNIX. *Prentice-Hall.*
(Stenback, 2003) Stenback, J. (2003). Document Object Model Level 3 Load and Save. *W3C Working Draft.*

(Stenback et al., 2003) Stenback, J., Le Hégaret, P., Le Hors, A., Wilson, C., Jacobs, I., Champion, M., Isaacs, S., Apparao, V. (2003). Document Object Model Level 2 HTML. *W3C Recommendation.*

(Stephen, 1979) Stephen C.J. (1979). YACC: Yet Another Compiler-Compiler. *Unix Programmer´s Manual Vol 2b.*

(Takahashi, 1975) Takahashi, M. (1975). Generalizations of Regular Sets and Their Application to a Study of Context-Free Languages. *Information and Control*, 27(1), pp. 1-36.

(Temprado-Battad et al., 2010) Tremprado-Battad, B., Sarasa-Cabezuelo, A., Sierra-Rodriguez, J.L.(2010). Modular Specifications of XML Processing Tasks with Attribute Grammars Defined on Multiple Syntactic Views. *20th International Workshop on Database and Expert Systems Application (DEXA'10)*. IEEE Computer Society, pp. 337-441.

(Temprado-Battad et al., 2011) Tremprado-Battad, B., Sarasa-Cabezuelo, A., Sierra-Rodriguez, J.L.(2011). Checking the conformance of grammar refinements with respect to initial context-free grammars. *Federated Conference on Computer Science and Information Systems (FedCSIS 2011)*. IEEE Computer Society, pp. 887-890.

(Thatcher, 1967) Thatcher, J.W. (1967).Characterizing Derivation Trees of Context-Free Grammars Through a Generalization of finite automata theory. *Journal of Computer and Systems Science*, 1(4), pp. 317-322.

(Thatcher, 1973) Thatcher, J.W. (1973). Tree Automata: An Informal Survey. *Currents in Theory of Computing*. Prentice-Hall, pp. 143–172.

(Thompson et al., 2001) Thompson, H.S, Beech, D., Maloney, M., Mendelsohn, N. (2001). XML Schema Part 1: Structures. *W3C Recommendation.*

(Tidwell, 2008) Tidwell, D. (2008). XSLT, 2nd Edition. *O'Reilly Media.*

(Tozawa, 2001) Tozawa, A. (2001). Towards Static Type Checking for XSLT. *ACM Symposium on Document Engineering(DocEng'01)*. ACM, pp. 18-27.

(Trancon et al., 2003) Trancón, B., Lepper, M., Wieland, J. (2003). Automatic Construction of XML-based Tools seen as Metaprogramming. *Automated Software Engineering, 10*, pp. 23-38.

(Vlist, 2001) Vlist, E. (2001). Comparing XML Schema Languages. *O'Reilly XML.com.*

(Vlist, 2003a) Vlist, E. (2003). Relax NG. *O'Relly Media.*

(Vlist, 2003b) Vlist, E. (2003). Examplotron V 0.7. *Technical Report.* Dyomedea.

(Vogt et al., 1989) Vogt, H. , Swierstra, S. D., Kuiper, M. F. (1989). Higher-Order Attribute Grammars. *ACM SIGPLAN 1989 Conference on Programming Language Design and Implementation (PLDI'89)*. ACM SIGPLAN Notices, 24(7), pp. 131-145.

(Vyk et al., 2006) Vyk, E.R.v. Schwerdfeger, A.C. (2006). Context-aware scanning for Parsing Extensible Languages. *6th International Conference on Generative Programming and Component Engineering (GPCE'06).* ACM*,* pp. 63-72.

(Wallmsley, 2007) Wallmsley, P. (2007). XQuery. *O'Reilly Media.*

(Watt, 1977) Watt, D.A. (1977). The Parsing Problem for Affix Grammars. *Acta Informatica*, 8, pp. 1-20.

(Whitmer, 2003) Whitmer,R. (2003).Document Object Model Level 3 XPath. *W3C Candidate Recommendation.*

(Wilhelm, 1982) Wilhelm, R. (1982). LL- and LR-Attributed Grammars. *Fachtagung über Programmiersprachen*. Informatik-Fachberichte. Springer, 53, pp. 151-164.

(Wilson et al., 2000) Wilson, C., Le Hégaret, P., Apparao, V. (2000). Document Object Model Level 2 Style. *W3C Recommendation.*

(Wyk et al., 2002) Wyk, E. V., Moor, O. D., Backhouse, K. (2002). Forwarding in Attribute Grammars for Modular Language Design. *11th International Conference on Compiler Construction*. Springer, pp. 128-142.

(Wyk et al., 2010) Wyk, E.V., Bodin, D., Gao, J., Krishnan, L. (2010). Silver: An Extensible Attribute Grammar System. *Science of Computer Programming*, 75(1-2), pp. 39-54.

(XMLPULL, 2004) XML Pull Parsing. (2004). *www.xmlpull.org/*

(XTutor, 2007) XTutor web site. (2007). *icampus.mit.edu/xtutor*.